



**Friedrich-Schiller-Universität Jena**  
Fakultät für Mathematik und Informatik  
Institut für Informatik

# **Repository-Mining von BPMN-Prozessen**

## **BACHELORARBEIT**

zur Erlangung des akademischen Grades  
Bachelor of Science (B. Sc.)  
im Studiengang Informatik

eingereicht von Viktor Stefanko  
geb. am 18.10.1991 in Serbytschany

Betreuer: Dr. Thomas Heinze  
Themenverantwortlicher: Prof. Dr. habil. Wolfram Amme

Jena, den 20. Juli 2019

## Zusammenfassung

Das Thema dieser Arbeit ist das Minen von öffentlichen GitHub-Repositorys nach BPMN-Prozessen. Es wurde das Mining-Programm entwickelt, mit dem ca. 2.103 GitHub-Repositorys pro Stunde durchsucht werden können. Dieses Programm kann parallel ausgeführt werden, was die Durchsuchung der GitHub-Repositorys erheblich beschleunigen kann.

Die parallele Anwendung des Mining-Programms auf 4 Rechner für die Durchsuchung 10% von allen öffentlichen GitHub-Repositorys hat 30 Tage, 12 Stunden und 48 Minuten gedauert. Es wurden insgesamt 6.163.217 GitHub-Repositorys durchsucht und bei 0,02% von ihnen (1.251 Repositorys) BPMN-Prozesse gefunden.

Die gefundene BPMN-Prozesse wurden nach verschiedenen Merkmalen, wie Dateialter, Anzahl der Dateiänderungen, Dateiautoren und Duplikate untersucht. Die Repositorys mit BPMN-Prozessen wurden ebenfalls nach Hauptprogrammiersprachen, Jahren der Erstellung und Standorten der Mitwirkenden untersucht. Bei diesen Analysen war deutlich, dass die Erscheinung der Version 2.0 von BPMN in 2011 ein großer Schub für die Verbreitung der BPMN-Prozesse auf GitHub war. Denn seit 2011 beträgt der jährliche Zuwachs der GitHub-Repositorys mit BPMN-Prozessen ganze 71,7%.

Die gefundenen BPMN-Prozesse im XML-Format wurden mithilfe des externen Programms BPMNspector nach Verstöße gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln analysiert. Es stellte sich heraus, dass die absolute Mehrheit der BPMN-Prozesse (83,5%) mit solchen Verstößen erstellt wurden.

# Inhaltsverzeichnis

<b>Zusammenfassung</b>	<b>3</b>
<b>Abkürzungsverzeichnis</b>	<b>5</b>
<b>1. Einleitung</b>	<b>6</b>
1.1. Einführung und Problemstellung . . . . .	6
1.2. Ziele . . . . .	6
1.3. Aufbau der Arbeit . . . . .	6
<b>2. Grundlagen</b>	<b>8</b>
2.1. BPMN . . . . .	8
2.2. Mining Software Repositorys . . . . .	15
<b>3. Mining von BPMN-Prozessen</b>	<b>18</b>
3.1. Ansätze . . . . .	18
3.1.1. GHTorrent . . . . .	18
3.1.2. Klonen . . . . .	19
3.1.3. GitHub API . . . . .	20
3.1.3.1. Suche via GitHub API . . . . .	20
3.1.3.2. Durchsuchen der Baumstruktur . . . . .	21
3.2. Implementierung des Mining-Prozesses . . . . .	22
3.2.1. Datenbank . . . . .	23
3.2.2. Das Mining-Programm . . . . .	25
3.3. Ergebnisse . . . . .	30
<b>4. Analyse gefundener BPMN-Dateien</b>	<b>33</b>
4.1. Allgemeine Statistiken . . . . .	33
4.1.1. Statistiken über GitHub-Repositorys . . . . .	33
4.1.1.1. Dominante Programmiersprache . . . . .	33
4.1.1.2. Erstellungsjahre und Jahre der letzten Änderung . .	34
4.1.1.3. Dauer der Aktivität . . . . .	36
4.1.1.4. Anzahl der Commits . . . . .	37
4.1.1.5. Standorte der Repository-Entwickler . . . . .	38
4.1.2. Statistiken über gefundene BPMN-Dateien . . . . .	39
4.1.2.1. Analyse der Dateien bezüglich Duplikate . . . . .	39
4.1.2.2. Anzahl der BPMN-Prozessen im XML-Format mit der grafischen Repräsentation . . . . .	40
4.1.2.3. Anzahl der XML-Elemente der BPMN-Prozesse . . .	42
4.1.2.4. Anzahl der Autoren . . . . .	42

4.1.2.5.	Dateialter . . . . .	43
4.1.2.6.	Anzahl der Dateiänderungen . . . . .	43
4.2.	Statistiken über Korrektheit der BPMN-Prozesse im XML-Format . .	44
4.2.1.	Analyse mittels BPMNspector . . . . .	44
4.2.2.	Reparieren mittels BPMNspector-fixSeqFlow . . . . .	47
<b>5.</b>	<b>Fazit</b>	<b>50</b>
	<b>Literaturverzeichnis</b>	<b>53</b>
	<b>Abbildungsverzeichnis</b>	<b>55</b>
	<b>Tabellenverzeichnis</b>	<b>56</b>
<b>A.</b>	<b>Anhang</b>	<b>57</b>
<b>B.</b>	<b>Anhang</b>	<b>59</b>
<b>C.</b>	<b>Anhang</b>	<b>65</b>

## Abkürzungsverzeichnis

<b>API</b>	Application Programming Interface
<b>BPE</b>	Business Process Engine
<b>BPM</b>	Business Process Managemen
<b>BPMN</b>	Business Process Model and Notation
<b>CSS</b>	Cascading Style Sheets
<b>EPK</b>	Ereignisgesteuerten Prozesskette
<b>EXT</b>	Extended Constraints
<b>GB</b>	Gigabyte
<b>HTML</b>	Hypertext Markup Language
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>ISO</b>	Internationale Organisation für Normung
<b>JSON</b>	JavaScript Object Notation
<b>KB</b>	Kilobyte
<b>MSR</b>	Mining Software Repositorys
<b>OMG</b>	Object Management Group
<b>SSH</b>	Secure Shell
<b>UML</b>	Unified Modeling Language
<b>XML</b>	Extensible Markup Language
<b>XSD</b>	XML Schema Definition

# 1. Einleitung

## 1.1. Einführung und Problemstellung

Die Business Process Model and Notation (BPMN) ist der führende Standard im Bereich der Geschäftsprozesse und Workflow-Modellierungssprachen [43]. Sie wurde im Jahre 2013 als Standard für die Prozessmodellierung von der Internationalen Organisation für Normung (ISO) festgeschrieben.

Die Benutzung der bestimmten Programmiersprachen oder Diagrammtypen in öffentlichen Software-Repositorys, wie z.B. GitHub, wird zurzeit von den IT-Spezialisten als Spiegelung der Tendenz in ganzer IT-Branche betrachtet. Obwohl die Bedeutung von BPMN für die Geschäftsmanagement und IT-Bereich seit der Standardisierung deutlich gewachsen wurde, gibt es bis jetzt noch keine wissenschaftliche Arbeiten, die die Nutzung von BPMN-Prozessdiagrammen in öffentlichen Software-Repositorys analysieren.

## 1.2. Ziele

Ziel von dieser Arbeit ist es, die öffentlichen Software-Repositorys auf GitHub nach BPMN-Prozesse zu durchsuchen und die gefundenen Dateien nach verschiedenen Merkmalen zu analysieren. Diese Arbeit sollte somit zeigen, sowohl, wie groß der Anteil der Repositorys mit BPMN-Prozessen ist, als auch, welche Korrelationen es zwischen Benutzung von BPMN-Prozessen und z.B. Programmiersprachen oder Standorten der GitHub-Benutzer gibt.

## 1.3. Aufbau der Arbeit

Diese Arbeit besteht aus folgenden Kapiteln:

### **Kapitel 2 - Grundlagen**

Hier werden Grundlagen über die BPMN-Prozesse und das Forschungsgebiet Mining Software Repositorys (MSR) präsentiert.

### **Kapitel 3 - Mining von BPMN-Prozessen**

Im diesen Kapitel werden zuerst Ansätze für die Suche nach BPMN-Prozessen auf GitHub analysiert, die Struktur des Mining-Programms beschrieben und anschließend die Ergebnisse der Programmausführung dargestellt.

**Kapitel 4 - Analyse gefundener BPMN-Prozesse**

Im vierten Kapitel werden gefundene BPMN-Prozesse analysiert. Hier werden zuerst allgemeine Statistiken über als BPMN-Prozesse erkannte Dateien und Repositories, zu denen sie gehören, dargestellt und anschließend Statistiken über Korrektheit der BPMN-Prozesse im XML-Format, die mithilfe des BPMNspectors gebildet wurden, präsentiert.

**Kapitel 5 - Zusammenfassung**

Hier, im letzten Kapitel, werden die Schlüsse der Arbeit gezogen.

## 2. Grundlagen

Das folgende Kapitel gibt einen kurzen Überblick über die Geschäftsprozessmodellierungssprache BPMN und das systematische Auswerten und Durchsuchen von Software Repositorys, welches als Forschungsgebiet unter dem Begriff Mining Software Repositorys (MSR) zusammengefasst wird.

### 2.1. BPMN

Die Business Process Model and Notation (BPMN) [10] ist eine Spezifikation, die eine grafische Notation zur Modellierung von Geschäftsprozessen und ein XML-Format zur Prozessserialisierung bereitstellt. Sie ist ein öffentlicher Standard, der von Object Management Group (OMG) [30] verwaltet wird. Das gesamte Dokument, das BPMN ausführlich beschreibt, umfasst mehr als fünf hundert Seiten und kann von der offiziellen Internetseite der OMG heruntergeladen werden. Siehe: [10].

Das Hauptziel von BPMN, wie in der Spezifikation [30] benannt, ist es, eine Notation bereitzustellen, die für alle Anwender leicht verständlich und zu verwenden ist. Dies beinhaltet sowohl die für die Prozessmodellierung verantwortlichen Domänenexperten, wie auch die IT-Experten, die die Prozesse im Anschluss implementieren. Somit schafft BPMN eine standardisierte Brücke für die Lücke zwischen Geschäftsprozessdesign und Prozessimplementierung [31].

Grundsätzlich basiert BPMN auf der Überarbeitung und Anpassung anderer Notationen und Methoden für die Geschäftsprozessmodellierung, insbesondere des Aktivitätsdiagramms der Unified Modeling Language (vereinheitlichte Modellierungssprache), kurz UML [44], des Aktivitäts-Entscheidungsflussdiagramms und der Ereignisgesteuerten Prozesskette (EPK) [33], [28]. Die aktuelle Version von BPMN (2.0.2) definiert unter anderem das XML-basierte Format für das Speichern und die Übertragung von BPMN-Prozessen.

Die BPMN-Prozesse können nach der Einführung vom XML-Format mithilfe von Business Process Engines (BPE) direkt ausgeführt werden [10]. Einer der zurzeit meist verbreiteter quelloffener BPE ist Camunda BPM [6], ein Java-basiertes System (Framework), das BPMN für die Workflow- und Prozessautomatisierung unterstützt. Dieses System wurde von in Berlin ansässigen Firma Camunda entwickelt und besteht aus einer Reihe von Komponenten. Eine von diesen Komponenten, Camunda Modeler [11], wird für die Erstellung und das Editieren von BPMN-Prozessen im XML-Format angewendet. Dieses Programm kann als Desktop-Anwendung oder als Eclipse plugin benutzt werden. Camunda Modeler



hat zwei Editoren: Grafischen- und Texteditor. Die Abbildungen 2.1 und 2.2 zeigen die beiden Editoren während der Erstellung eines BPMN-Prozessdiagramms, das am Ende dieses Abschnitts in der Abbildung 2.3 präziser dargestellt und beschrieben ist.

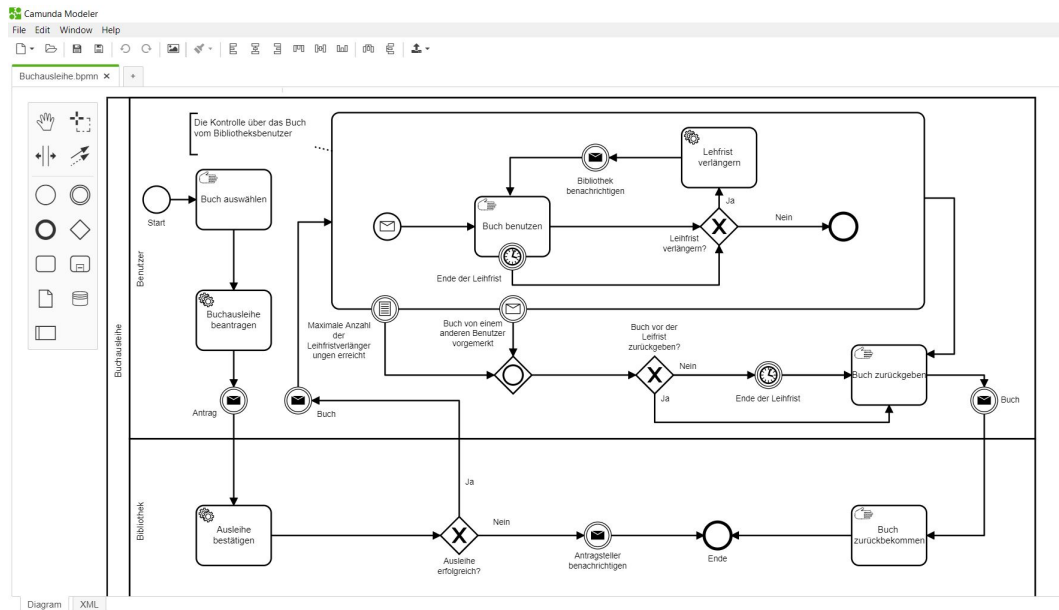


Abbildung 2.1.: Camunda Modeler (grafischer Editor)

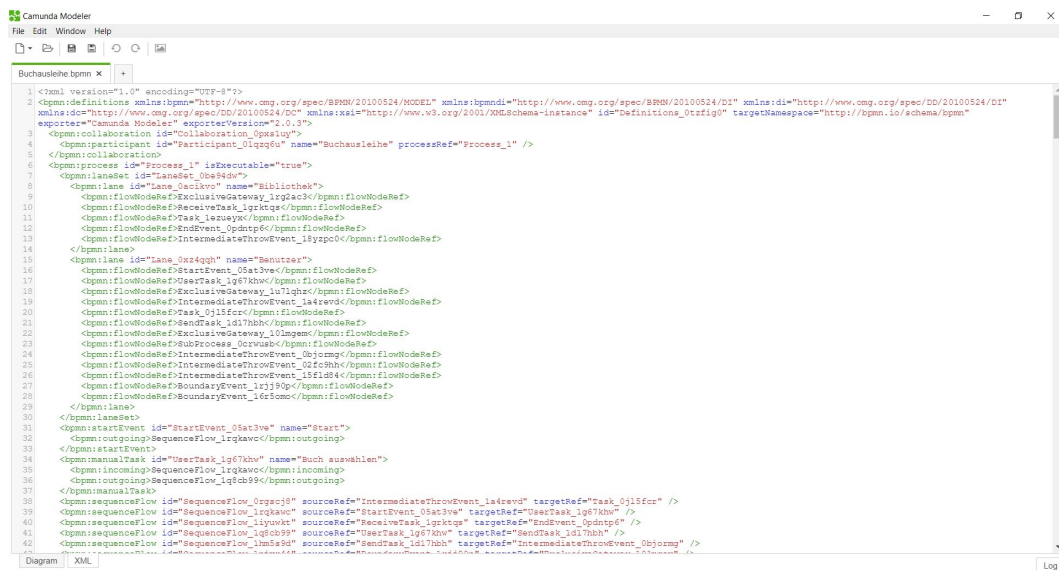


Abbildung 2.2.: Camunda Modeler (Texteditor)

Insgesamt werden in der Spezifikation von BPMN vier grundlegende Arten von Prozessstypen definiert:

- Prozessdiagramm (Process Diagram)
- Kollaborationsdiagramm (Collaboration Diagram)
- Choreographie-Diagramm (Choreography Diagram)
- Konversationsdiagramm (Conversation Diagram)[31].

Für die Definition eines Prozesses, stellt die Spezifikation von BPMN eine Men-

ge von Elementen sowie Regeln für deren Verwendung bereit. Derart werden die Syntax und Semantik von BPMN natürlichsprachlich definiert [29]. Jedes BPMN-Element kann dabei einer von fünf grundsätzlichen Elementkategorien zugeordnet werden [10]. Diese sind:

1. **Flow Objects** (Knoten)
2. **Data** (Daten)
3. **Connecting Objects** (Verbindende Objekte)
4. **Swimlanes** (Schwimmbahnen, die Teilnehmer darstellen)
5. **Artifacts** (Artefakte)

In der Tabelle 2.1 werden die wichtigsten Elemente von BPMN-Prozessen kurz erläutert sowie ihre visuelle Notation angegeben.

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen [10], [29], [1]

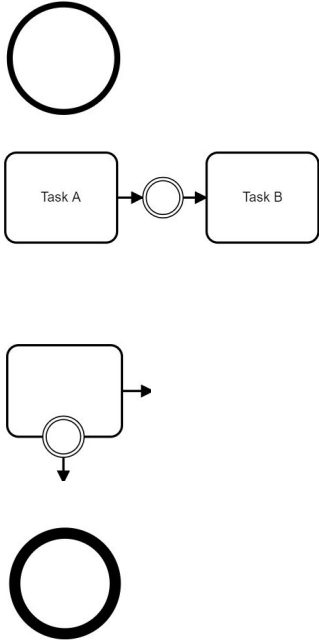
Element	Beschreibung	Grafische Notation
Event (Ereignis)	<p>Ereignisse geben an, wann- und was für ein Ereignis auftritt. Die Ereignisse können in 4 Kategorien eingruppiert werden, je nachdem, wann sie den Prozessablauf beeinflussen:</p> <ul style="list-style-type: none"> <li>• <b>Startereignis</b> (Start Event) bezeichnet den Auslöser eines Prozesses.</li> <li>• <b>Zwischenereignisse</b> (Intermediate Events) zeigen Zustandsänderungen im Prozess an.</li> <li>• <b>Grenzereignisse</b> (Border Events) treten während der Abarbeitung einer Aktivität auf und ändern den Prozessfluss.</li> <li>• <b>Endereignis</b> (End Event) bezeichnet das Ende eines Prozesses.</li> </ul> <p>Der Typ des Ereignisses wird durch ein Symbol in der Mitte des Kreises bestimmt, siehe Abbildung 2.3.</p>	

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen








Element	Beschreibung	Grafische Notation
Activity (Aktivität)	<p>Eine Aktivität ist ein Oberbegriff für die Arbeitsschritte, die in einem Prozess ausgeführt werden. Eine Aktivität wird als Rechteck mit abgerundeten Ecken gezeichnet und kann atomar oder zusammengesetzt sein. Eine Atomare Aktivität ist eine Aufgabe (Task), die beschreibt, was zu erledigen ist. Eine zusammengesetzte Aktivität ist ein Unterprozess (Subprocess), der aus anderen BPMN-Elementen bestehen kann.</p> <p>Zu den meistverbreiteten Aufgabentypen (Tasks) gehören folgende:</p> <ul style="list-style-type: none"> <li>• <b>Erhalten der Nachricht</b> (Receive Task) ist eine Alternative zu einem Ereignis, das das Bekommen einer Nachricht modelliert.</li> <li>• <b>Skript Aufgabe</b> (Script Task) bezeichnet Ausführung eines Skriptes vom BPE. Die Programmiersprache, in dem das Skript geschrieben ist, muss BPE bekannt sein.</li> <li>• <b>Service Aufgabe</b> (Service Task) bezeichnet eine Aufgabe, die vom einem Programm während der Prozessausführung erledigt wird.</li> <li>• <b>Manuelle Aufgabe</b> (Manual Task) bezeichnet eine Aufgabe, die von einem Menschen zu erledigen ist.</li> <li>• <b>Benutzer-Aufgabe</b> (User Task) ähnelt der manuellen Aufgabe, hier weist BPE die Aufgaben aber den konkreten Personen zu.</li> <li>• <b>Senden-Aufgabe</b> (Send Task) ist eine technische Aufgaben, die vom BPE ausgeführt werden. Zum Beispiel ein synchroner Aufruf von einem Webservice.</li> </ul>	            

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen







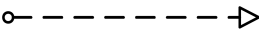
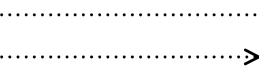


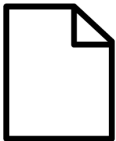

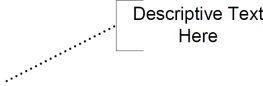
Element	Beschreibung	Grafische Notation
Gateway (Entscheidungs- punkt)	<p>Ein Entscheidungspunkt wird verwendet, um die Divergenz und Konvergenz von Sequenzflüssen zu steuern. Es gibt fünf Arten von Entscheidungspunkten:</p> <ul style="list-style-type: none"> <li>• <b>Datenbasiertes exklusive Gateway</b> (XOR-Gateway) lässt genau einen Prozesspfad zu.</li> <li>• <b>Paralleles Gateway</b> (AND-Gateway) alle Pfade müssen durchschritten werden.</li> <li>• <b>Datenbasiertes inklusive Gateway</b> (OR-Gateway) es muss mindestens ein Pfad gewählt werden.</li> <li>• <b>Ereignisgesteuertes Gateway</b> (Event-Gateway). Es wird wie beim XOR-Gateway nur ein Pfad gewählt. Dieser Pfad wird aber nicht anhand von Daten, sondern anhand des nächsten Ereignisses bestimmt.</li> <li>• <b>Komplexes Gateway</b> (Complex-Gateway) wird zum Modellieren des komplexen Synchronisationsverhaltens verwendet.</li> </ul>	    
Sequence Flow (Sequenzfluss)	Ein Sequenzfluss wird verwendet, um die Reihenfolge zu definieren, in der Aktivitäten ausgeführt werden.	
Message Flow (Nachrichtenfluss)	Ein Nachrichtenfluss wird verwendet, um den Nachrichtenfluss zwischen Prozessteilnehmern zu modellieren.	
Association (Assoziation)	Eine Assoziation wird verwendet, um Informationen und Artefakte, beispielsweise Datenobjekte, mit anderen BPMN-Elementen, beispielsweise Aktivitäten, zu verknüpfen.	

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen

Element	Beschreibung	Grafische Notation
Pool (Schwimmbecken)	Ein Pool ist die grafische Darstellung eines Teilnehmers an einer Kollaboration. Er wird auch als Swimlane bezeichnet und fungiert als grafischer Container, um eine Gruppe von Aktivitäten von anderen Pools zu trennen.	
Lane (Spur)	Eine Spur ist eine Unterpartition innerhalb eines Prozesses, manchmal innerhalb eines Pools, und erstreckt sich über die gesamte Länge des Prozessdiagramms, entweder vertikal oder horizontal.	
Data Object (Datenobjekt)	Datenobjekte geben an, welche Informationen von Prozessaktivitäten produziert und/oder konsumiert werden. Datenobjekte können zwischen Aktivitäten ausgetauscht werden und definieren dadurch Datenflüsse im Prozess.	
Message (Nachricht)	Eine Nachricht wird verwendet, um den Austausch von Informationen zwischen zwei Teilnehmern, typischerweise über Prozessgrenzen hinweg, darzustellen. Der Nachrichtenaustausch kann dabei sowohl synchron als auch asynchron erfolgen.	
Text Annotation (Textanmerkung)	Textanmerkungen sind ein Mechanismus für einen Modellierer, um dem Leser eines BPMN-Diagramms zusätzliche Textinformationen, beispielsweise Kommentare, bereitzustellen.	

Wir möchten uns in der Abbildung 2.3 ein einfaches BPMN-Prozess anschauen, das den Prozess der Buchausleihe aus einer Bibliothek von einem Benutzer modelliert.

Bei dem Prozess „Buchausleihe“ sind Bibliothek als Institution und ein Bibliotheksbenutzer beteiligt. Der Prozess wird vom Benutzer gestartet, der zuerst ein Buch für die Ausleihe auswählt und Buchausleihe beantragt. Die Bibliothek kann die Ausleihe bestätigen oder absagen. Im Falle der Bestätigung wird das Buch ausgeliehen und im Falle der Absage wird dem Antragsteller der Grund der Absage mitgeteilt.

Der Benutzer kann das Buch vor dem Leihfrist oder am Ende der Leihfrist der Bibliothek zurückgeben. Es ist auch möglich, die Leihfrist zu verlängern. Das wird aber nur dann erlaubt, wenn die maximale Anzahl der Leihfristverlängerungen nicht erreicht wurde und das Buch von keinem anderen Bibliotheksbenutzer vorgemerkt wurde. Falls eine von diesen Bedingungen nicht zutrifft, kann die Leihfrist nicht mehr verlängert werden und das Buch wird vor dem Leihfrist oder am Ende der Leihfrist der Bibliothek zurückgegeben. Der Prozess ist zu Ende, wenn das

Buch der Bibliothek zurückgegeben wird.

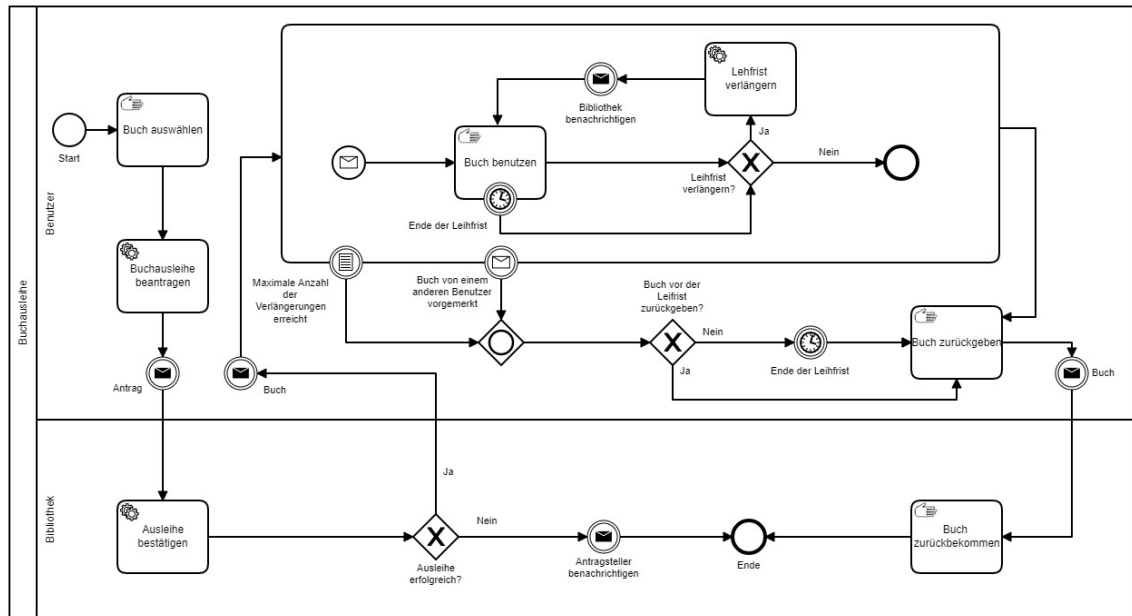


Abbildung 2.3.: Buchausleihe

Jetzt versuchen wir die Abbildung 2.3 zu beschreiben und dabei die dargestellten Elemente zu benennen. Insgesamt besteht dieses Diagramm aus folgenden Elementen:

- Schwimmbecken (Pool).
- Spur (Lane).
- Startereignis (Start Event).
- Zwischenereignisse (Intermediate Events): Message Catch Event, Message Throw Event, Timer Catch Event.
- Grenzerereignisse (Border Events): Conditional Catch Event, Message Catch Event, Timer Catch Event.
- Aktivitäten (Activities): ein Unterprozess (Subprocess) und Aufgaben (Tasks). Aufgabentypen: Manuelle Aufgabe (Manual Task) und Service Aufgabe (Service Task).
- Gateways: XOR-Gateway und OR-Gateway.
- Sequenzfluss (Sequence Flow).
- Endereignis (End Event).

Der gesamte Prozess befindet sich innerhalb des Pools und hat den Namen „Buchausleihe“. Der Pool ist in zwei Bahnen (eng. „Lanes“) geteilt. Die Teilung in Bahnen macht es deutlich, welche Aktivitäten von der Bibliothek als Institution und welche von dem Bibliotheksbenutzer gemacht werden. Der Prozess startet mit dem Start Event. Die vom Start Event ausgehende Kante (Sequence Flow) zeigt, welches Element als Nächstes durchlaufen wird. So werden zuerst die Tasks „Buch auswählen“, „Buchausleihe beantragen“ von Bibliotheksbenutzer und „Ausleihe bestätigen“ von der Bibliothek erledigt. Danach wird ein XOR-Gateway passiert,

das prüft, ob die Buchausleihe bestätigt wurde. Im Falle der Absage wird Message Throw Event ausgelöst und den Grund der Absage dem Antragsteller mitgeteilt. Im Falle der Zusage bekommt der Antragsteller das von ihm ausgeliehene Buch.

Die Kontrolle über das Buch von dem Bibliotheksbenutzer ist in der Abbildung 2.3 als ein Unterprozess dargestellt, der startet, sobald das Buch ausgeliehen wird. So kommen wir zum Task „Buch benutzen“. Dieser Task hat ein Grenzereignis (Timer Catch Event), das passiert, wenn die Leihfrist endet. In diesem Fall wird ein XOR-Gateway erreicht, wo der Benutzer die Leihfrist verlängern oder das Buch der Bibliothek zurückgeben kann. Dieses XOR-Gateway kann auch über eine direkte Kante vom „Buch benutzen“ Task erreicht werden. In diesem Fall entscheidet der Benutzer über die Zurückgabe des Buchs oder über die Leihfristverlängerung vor der geltenden Leihfrist. Falls der Benutzer die Leihfrist verlängern will, wird das „Leihfrist verlängern“ Task abgearbeitet und die Bibliothek über die Leihfristverlängerung benachrichtigt (Message Throw Event). So wird das „Buch benutzen“ Task erreicht und der beschriebene Vorgang wiederholt sich. Falls der Benutzer sich für die Zurückgabe des Buchs entscheidet, endet dieser Unterprozess und das „Buch zurückgeben“ Task wird erreicht.

Der oben beschriebene Unterprozess kann von einer der zwei Grenzereignissen unterbrochen werden: beim Bekommen einer Nachricht, dass das Buch von einem anderen Bibliotheksbenutzer vorgemerkt wurde (Message Catch Event) oder wenn die maximale Anzahl der Leihfristverlängerungen erreicht wurde (Conditional Catch Event). In diesem Fall wird der Unterprozess unterbrochen und das Buch vor/beim Erreichen der Leihfrist der Bibliothek zurückgegeben („Buch zurückgeben“ Task). Der gesamte Prozess endet, sobald die Bibliothek das ausgeliehene Buch zurückbekommt.

In der Abbildung fällt auf, dass es zwei Typen von Tasks benutzt wurden: Manual- und Service Tasks. Beim Bezeichnen eines Tasks als „Manual“ wurde angenommen, dass dieses „manuell“ von einer Person erledigt wird, z.B. das „Buch auswählen“ Task. Bei Service Task wurde angenommen, dass die Bibliothek über ein Webservice verfügt, über das die Ausleihe von der Bibliothek bestätigt oder vom Benutzer verlängert werden kann.

## 2.2. Mining Software Repositorys

Das Forschungsgebiet Mining Software Repositorys (MSR) analysiert die heute in Software-Repositorys, wie beispielsweise GitHub, vielfältig vorliegenden Daten. Auf diese Weise lassen sich interessante Rückschlüsse auf Softwareentwicklungsprozesse ziehen. Zudem können Aussagen zu Prozessen und der aus diesen resultierenden Software empirisch untersucht und belegt werden.

Beispiele für Software-Repositorys sind:

- **Historische Repositorys** wie Quellcodeverwaltungs-Repositorys, Fehler-Repositorys und archivierte Kommunikationen zeichnen verschiedene In-

formationen über die Entwicklung und den Fortschritt eines Repository auf [25]. Die Quellcodeverwaltungs-Repositorys, wie beispielsweise Git-Versionsverwaltungssysteme GitHub [19] und GitLab [22], enthalten den Quellcode verschiedener Anwendungen, die von mehreren Entwicklern geschrieben wurden.

- **Laufzeit-Repositorys** wie Bereitstellungsprotokolle enthalten Informationen zur Ausführung und Verwendung einer Anwendung an einer oder mehreren Bereitstellungsstandorten [25].

Programmierer erstellen und sammeln während der Softwareentwicklung viele Daten, die alle automatisch abgerufen und analysiert werden können. Wie zum Beispiel:

- Der Quellcode des Programms. Dies ist die wichtigste Eingabe für die mögliche Analyse.
- Durch die während der Ausführung der Software gesammelte Daten können Profile erhalten werden, die mitteilen, welche Teile der Software häufig verwendet werden und welche nicht.
- Den Produkten liegen möglicherweise zusätzliche Dokumentationen bei, beispielsweise: Wikis oder Issue Tracker (siehe GitHub/GitLab). Diese können auch wichtige Funktionen enthalten, die erklären, warum Code so aussieht, wie er aussieht.
- Die resultierende Software kann statisch analysiert werden und bietet Funktionen wie Komplexitätsmetriken oder Abhängigkeiten. Die Ergebnisse von solchen statischen Analysen, aber auch anderen Analysen und Tests, finden sich oft in Continuous Integration Systemen (CI-Systemen) wieder.
- Versionsarchive zeichnen die am Produkt vorgenommenen Änderungen auf, einschließlich wer, wann, wo und warum die Änderungen durchgeführt hat [27].

Die Abbildung 2.4 zeigt den allgemeinen Algorithmus für das MSR in Form eines BPMN-Prozesses.

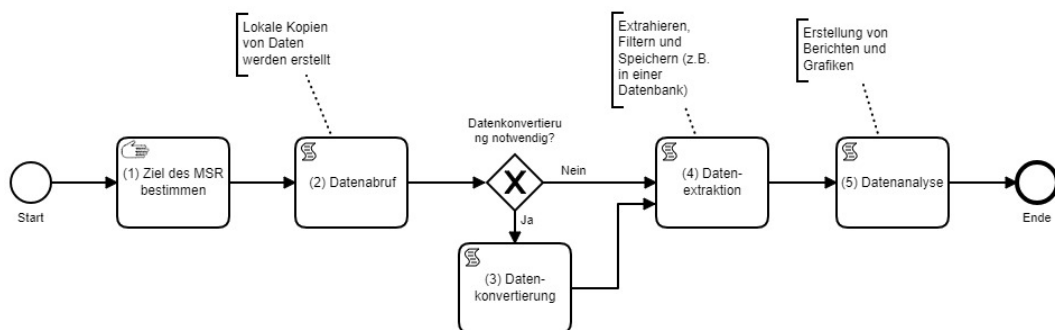


Abbildung 2.4.: Algorithmus für das MSR

1. **Ziel des MSR bestimmen**, d.h. welche Daten sind zur Beantwortung der Fragestellungen oder zur Validierung der Hypothesen notwendig.



2. **Datenabruf.** Um die ständige Verfügbarkeit der gezielten Daten sicherzustellen und einen schnellen Zugriff auf sie zu ermöglichen, sollten lokale Kopien erstellt werden.
3. **Datenkonvertierung** (optional). Data Mining erfordert, dass die Daten nicht nur heruntergeladen und verarbeitet werden, sondern auch, dass man viele ihrer Merkmale im Voraus versteht. Daher sollten die Daten den Voraussetzungen und Annahmen entsprechen, die vorher getroffen wurden.
4. **Datenextraktion.** Nachdem die Daten auf lokaler Festplatte in einer geeigneten Form gesichert wurden, können Sie verarbeitet werden. Die Verarbeitung umfasst das Extrahieren, Filtern und Speichern der Inhalte der Ressourcen in einem dauerhaften und für den Menschen lesbaren Format. Zum Beispiel können die gefilterten Informationen in einem relationalen Datenbankmanagementsystem gespeichert werden.
5. **Datenanalyse.** In diesem Schritt wird die gezielte Analyse der gespeicherten Daten durchgeführt [27].

## 3. Mining von BPMN-Prozessen

Das folgende Kapitel erläutert Ansätze für die Suche nach BPMN-Prozessen auf GitHub, die Struktur des Mining-Programms und Ergebnisse der Programmausführung.

### 3.1. Ansätze

In der Vorbereitungsphase wurden verschiedene Ansätze für die systematische Suche nach GitHub-Repositorys, die BPMN-Prozesse enthalten, untersucht und getestet. Grundlage und Inspiration dafür bildeten Publikationen zum Repository-Mining von UML-Diagrammen [15], [26], [38], [37].

#### 3.1.1. GHTorrent

Entsprechend der offiziellen Quellen von GitHub verfügt diese Plattform weltweit über mehr als 31 Millionen registrierte Benutzer und mehr als 100 Millionen Repositorys [20], wobei fast ein Drittel von ihnen innerhalb des letzten Jahres erzeugt wurden [41]. Diese Zahlen zeigen, dass GitHub eine sehr dynamische Hosting-Plattform ist und einen sehr großen Bestand von Daten enthält. Um diese Daten effektiv analysieren zu können, wird eine statische, zuverlässige und aktualisierte Quelle dieser Plattform benötigt [15]. Als solche Quelle wurde von uns das GHTorrent-Projekt ausgewählt [40].

Das GHTorrent-Projekt wurde entwickelt, wie seine Gründer angeben, um das Mining von GitHub-Repositorys zu erleichtern. Dieses Projekt bietet einen skalierbaren, abfragbaren Offline-Spiegel der GitHub-Daten, die auch durch die GitHub API angeboten werden [24]. GHTorrent aktualisiert seine Daten jeden Monat. Die Daten können sowohl heruntergeladen als auch ohne herunterzuladen online abgefragt werden. Herunterladen kann man sie als MySQL-Dump (Menge von csv-Dateien, wobei jede von denen einer MySQL-Datenbanktabelle entspricht) oder als MongoDB-Dump. Das Abfragen ist möglich sowohl durch eine GHTorrent-Online-Abfrageoberfläche [24] oder mithilfe von Google BigQuery [23], das einen aktuellen Import des neuesten MySQL-Dumps von GHTorrent enthält [18].

Nachdem der MySQL-Dump als tar-Archiv (85,5 GByte) mit 23 unterschiedlichen csv-Dateien und MySQL-Datenbankschema heruntergeladen wurde, haben wir die Dateien „users.csv“, die Informationen über GitHub-Benutzer enthält, und „projects.csv“ mit Informationen über GitHub-Repositorys lokal in gleichnamige Datenbanktabellen importiert (siehe auch Abbildung 3.1). Auf Grundlage dieser Da-

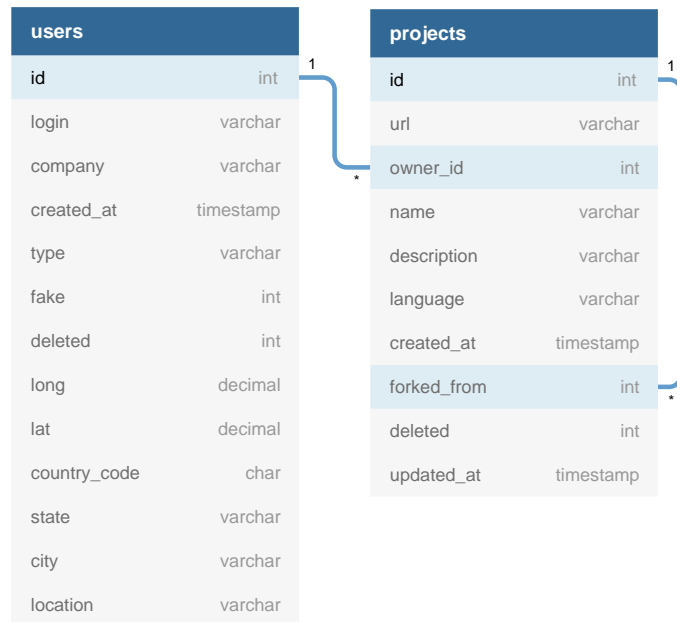


Abbildung 3.1.: MySQL-Datenbankschema der Tabellen „users“ und „projects“

ten konnten wir eine List von nicht-gelöschten und nicht-geforkten Repositories bilden.

GHTorrent stellt allerdings keine Informationen zu Dateien aus einem bestimmten GitHub-Repository zur Verfügung [15]. Es ist daher erforderlich, diese Informationen entweder über das Klonen und anschließende lokale Durchsuchen der Repositories oder online über Abfragen der GitHub API abzurufen.

### 3.1.2. Klonen

Da GitHub die verteilte Versionskontrolle `Git` als Standardmechanismus für die Verwaltung von Softwaresystemen verwendet, war es möglich mithilfe des `git clone` Befehls, die GitHub-Repositories herunterzuladen und dann nach BPMN-Prozessen zu durchsuchen.

Es wurde ein Python-Programm für die Suche nach Repositories mit BPMN-Prozessen geschrieben. In diesem Programm wurde in einer Schleife über die Liste mit Namen der GitHub-Repositories iteriert und bei jeder Iteration mit `git clone` ein Repository heruntergeladen und durchsucht.

Trotz der richtigen Ergebnissen, die dieses Verfahren liefert, ist es zu langsam, wenn es sich um große Repositories handelt. Dieses Problem wird auch in [39] beschrieben. Das Problem besteht darin, dass beim Klonen eines Repository nicht nur Dateien, sondern auch die ganze Geschichte des Repository heruntergeladen wird. Bei großen Repositories werden bis zu mehreren GB Daten heruntergeladen, was die entsprechende Ressourcen benötigt. Deshalb stellte sich dieses Verfahren für unsere Forschung als ineffektiv heraus.

### 3.1.3. GitHub API

GitHub bietet eine Anwendungsprogrammierschnittstelle (englisch: Application Programming Interface, kurz API) für den Zugriff auf Daten in öffentlichen Repositories. Der gesamte API-Zugriff erfolgt über HTTPS und kann über `https://api.github.com` aufgerufen werden. Es handelt sich dabei um eine REST-Schnittstelle und alle Daten werden als JSON-Objekte gesendet und empfangen [35].

Die GitHub API definiert eine Beschränkung von 5000 Anfragen pro Stunde und Konto für authentifizierte Benutzer und 60 Anfragen pro Stunde für nicht authentifizierte. Nicht authentifizierte Anfragen werden der IP-Adresse zugeordnet und nicht dem Benutzer, der Anfragen stellt [35].

#### 3.1.3.1. Suche via GitHub API

Die Suchschnittstelle der GitHub API ermöglicht die Suche nach Objekten in Repositories. Es können, beispielsweise, spezifische Daten (z. B. Code, Sterne, Commits oder Mitwirkende) in einem oder in mehreren Repositories gesucht werden. Allerdings gibt es folgende Beschränkungen bei der Such-API:

- Es können höchstens 30 authentifizierte oder 10 nicht authentifizierte Anfragen pro Minute gestellt werden.
- Die Antwort für eine Anfrage kann höchstens 1000 Resultate haben.
- Es wird nur der Standardzweig eines Repository (default branch) bei der Suche berücksichtigt. In den meisten Fällen ist dies der Hauptzweig (master branch).
- Nur Dateien, die kleiner als 384 KB sind, können durchsucht werden [35].

Es wurde wiederum ein Python-Programm geschrieben, in dem in einer Schleife über die Liste mit Namen der GitHub-Repositories iteriert wurde und bei jeder Iteration eine Anfrage über die GitHub API gesendet wurde.

Zum Beispiel für die Suche nach dem Schlüsselwort `bpmn` im Dateiinhalt wurde für GitHub-Benutzer `ViktorStefanko` und Repository `BPMN_Crawler` folgende Anfrage gesendet:

```
https://api.github.com/search/code?q=bpmn+in:file+user:ViktorStefanko+repo:BPMN_Crawler.
```

Die Abbildung 3.2 zeigt das zurückgelieferte JSON-Objekt. Man kann sehen, dass es innerhalb des `BPMN_Crawler` Repository 6 Dateien gefunden wurden, die das Schlüsselwort `bpmn` im Dateiinhalt enthalten.

Die Suche via GitHub API hatte wichtige Vorteile im Vergleich zum oben beschriebenen Vorgehen über das Klonen von Repositories. Erstens werden die Pfade zu allen Dateien die das gesuchte Wort enthalten in Form eines JSON-Objekts zurückgeliefert. Damit erhalten wir die Information, welche Dateien für das Projekt zu extrahieren sind. Zweitens ist der Ansatz im Vergleich zum Klonen von Reposito-

```

1  {
2    "total_count": 6,
3    "incomplete_results": false,
4    "items": [
5      {"name": "run_bpmn_spector.py"...},
61     {"name": "file_statistics.py"...},
157    {"name": "tree_crawler.py"...},
233    {"name": "README.md"...},
309    {"name": "schema.sql"...},
385    {"name": "main_script.py"...}
461  ]
462 }
```

Abbildung 3.2.: Beispiel von einem zurückgelieferten JSON-Objekt von Suchschnittstelle der GitHub API

rys schneller.

Allerdings gibt es auch Nachteile im Vergleich zum Klonen. Zum einen können Dateien größer als 384 KB nicht durchsucht werden. Und zum anderen können nur 1000 Resultate für ein Repository ermittelt werden. Folglich können bei sehr großen Repositories nicht alle Ergebnisse ermittelt werden.

Insgesamt lassen sich mit Hilfe der Suchschnittstelle der GitHub API 1800 Repositories pro Stunden durchsuchen (30 authentifizierte HTTP-Anfragen pro 1 Minute \* 60 Minuten = 1800).

### 3.1.3.2. Durchsuchen der Baumstruktur

Es gab noch ein Verfahren, dass das Benutzen der GitHub API möglich machte. Dessen Besonderheit ist es, dass nicht der Inhalt einer Datei in dem entsprechenden Projekt nach Schlüsselwörtern durchsucht wird, sondern es wird zuerst die Repositorystruktur als Baum mit Dateinamen und Ordnern bestimmt und nur in Dateinamen einschließlich der Dateiendung nach Schlüsselwörtern gesucht [15].

Dieses Verfahren besteht aus drei Schritten:

- Ermitteln des letzten commit, das dem Standardzweig eines Repository gehört. Dafür wird eine authentifizierte HTTP-Anfrage an die GitHub API mit (branches/master) als Parameter gesendet. Falls master branch der Standardzweig, also default branch ist, enthält das Ergebnis der Anfrage das letzte commit, was das Ziel war.

Es kann allerdings vorkommen, dass master branch nicht der Standardzweig (default branch) ist. In diesem Fall wird die zweite Anfrage gesendet, um den Standardzweig zu ermitteln. Es sei dies der Standardzweig defBr, dann enthält die dritte Anfrage die gesendet wird (branches/defBr) als Parameter. So wird auch hier das letzte commit ermittelt.

- Senden einer authentifizierten HTTP-Anfrage, um die Baumstruktur des jeweiligen Repository anhand des letzten `commit` zu erhalten.
- Suchen in Blättern des Baums, die Dateinamen des jeweiligen Repository sind, nach Schlüsselwörtern. Falls gefunden, Speichern des Dateipfads in einer Datenbank.

Dieses Verfahren verbindet die Vorteile des Klonens von Repositories und der Suche via GitHub API. Somit es ist schnell und es werden alle Repositories und ihre Dateinamen unabhängig von ihren Größen durchsucht.

Wegen der Beschränkung der GitHub API, können bei diesem Ansatz höchstens 5000 authentifizierte HTTP-Anfragen gestellt werden. Für jedes Repository werden mindestens zwei Anfragen gestellt: erste fürs Ermitteln des letzten `commit` und zweite fürs Ermitteln der Baumstruktur. Es können aber zusätzlich noch zwei Anfragen gebraucht werden. Das ist dann der Fall, wenn der Hauptzweig nicht `master branch` ist. Aus diesen Gründen können bei diesem Verfahren theoretisch mindestens 1250 und höchstens 2500 GitHub-Repositories pro Stunde durchsucht werden.

Bei der Umsetzung des Ansatzes stellte sich heraus, dass die Mehrheit von GitHub-Repositories `master branch` als Standardzweig hat. Daher war die Anzahl der GitHub-Repositories, die pro Stunde durchsucht werden konnten, nah zur oberen Schranke mit 2500 GitHub-Repositories. Somit war dieser Ansatz auch in diesem Sinne effektiver als das Klonen und die Suche via GitHub API.

## 3.2. Implementierung des Mining-Prozesses

Nach der Analyse verschiedener Ansätze haben wir GHTorrent, als Quelle für grundlegende Informationen zu GitHub-Repositories wie Name und Status ausgewählt. Zusätzlich konnte über diese Ressource auch der GitHub-Nutzer, der das Repository angelegt hat, bestimmt werden. Weil GHTorrent aber keine Informationen zu den in einem Repository enthaltenen Dateien liefert, wurde zusätzlich der in Abschnitt 3.1.3.2 beschriebene Ansatz zum Durchsuchen der Repositorybaumstruktur mit Hilfe der GitHub API verwendet.

Die Arbeit am Programm startete am Anfang des Novembers 2018. Es wurde der GHTorrent MySQL-Dump von 01.11.2018 heruntergeladen und daraus die MySQL-Datenbanktabellen „users“ und „projects“ (siehe Abbildung 3.1) extrahiert. Diese zwei Tabellen enthalten Daten, die es möglich machen, eine Liste von nicht-gelöschten und nicht-geforkten GitHub-Repositories zu bilden. Die SQL-Anfrage dafür sieht wie folgt aus:

```
SELECT users.login, projects.name FROM users, projects
WHERE projects.owner_id=users.id AND projects.deleted=0 AND
projects.forked_from is NULL.
```

Das Ergebnis dieser Anfrage liefert die Namen und Zugangsdaten zu 61.632.173 GitHub-Repositories. Wegen der Beschränkung der GitHub API, die höchstens 5000

authentifizierte HTTP-Anfragen pro Stunde erlaubt, würde das Durchsuchen dieser GitHub-Repositorys im besten Fall (siehe obere Schranke in 3.1.3.2) 1.027 Tage dauern. Da dies nicht umsetzbar war, wurde die Entscheidung getroffen, nur 10 Prozent von der gesamten Anzahl, d.h. 6.163.217 GitHub-Repositorys, zu durchsuchen. Diese Anzahl wiederum konnte im besten Fall in 103 Tage durchsucht werden.

Um die Durchsuchung der GitHub-Repositorys zu beschleunigen, sollte das Programm so entwickelt werden, dass es auf einem Rechner mehrfach gestartet werden kann. Es sollte also so programmiert werden, dass mehrere Instanzen des Programms mit unterschiedlichen Zugangsdaten initialisiert werden und gleichzeitig unterschiedliche Repositorys durchsucht werden.

### 3.2.1. Datenbank

Die Abbildung 3.3 zeigt uns die Struktur der verwendeten Datenbank, die als SQLite-Datenbank implementiert wurde.

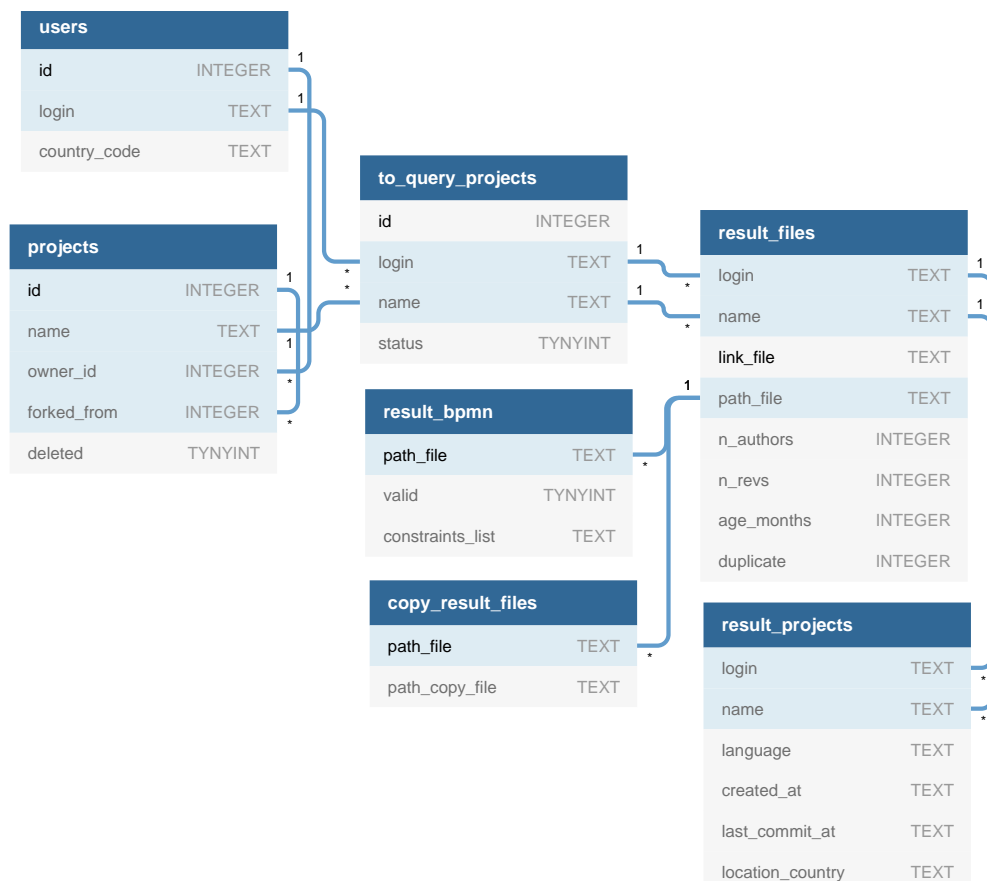


Abbildung 3.3.: Datenbankschema

Die **users** Tabelle enthält Informationen über GitHub-Nutzer und zwar Zugangsdaten (*login*) und Ländercode (*country\_code*).

Die Tabelle **projects** beschreibt ein GitHub-Repository. Sie enthält Name des Repository (*name*), Verweis auf Inhaber (*owner\_id*) in Tabelle **users**, Attribut *forked\_from*, in dem gespeichert wird, von welchem Repository es abgeleitet wurde, wenn das

überhaupt der Fall ist, und Attribut *deleted*, das angibt, ob das Repository gelöscht wurde.

Die Inhalte der **users** und **projects** Tabellen wurden aus gleichnamigen Tabellen der MySQL-Datenbank, die aus dem GHTorrent-Archiv stammt, siehe auch Abbildung 3.1.

Die Tabelle **to\_query\_projects** dient als Eingabequelle für das Mining-Programm. Sie enthält Zugangsdaten (*login*) und Name (*name*) von 10 Prozent der nicht-gelöschten und nicht-geforkten GitHub-Repositorys. Im Attribut *status* wird gespeichert, ob das Repository durchsucht wurde (Wert 1) oder noch zu durchsuchen ist (Wert 0). Die Daten wurden in diese Tabelle randomisiert aus **users** und **projects** mit folgender Anfrage eingefügt:

```
INSERT INTO to_query_projects (login, name) SELECT
users.login, projects.name FROM users, projects WHERE
projects.owner_id=users.id AND projects.deleted=0 AND
projects.forked_from is NULL ORDER BY RANDOM() LIMIT 6163217.
```

Die Tabelle **result\_files** enthält Informationen über Dateien, die das Mining-Programm als potentielle BPMN-Prozesse eingestuft hat. Sie enthält folgende Attribute: Zugangsdaten des Inhabers (*login*), Name vom GitHub Repository (*name*), Weblink (*link\_file*), Pfad innerhalb des Repository (*path\_file*), Anzahl der Autoren (*n\_authors*) und Änderungen (*n\_revs*), Alter der Datei in Monaten (*age\_months*) und Attribut *duplicate*, das angibt, ob Duplikate der entsprechenden Datei gefunden wurden.

Die Tabelle **result\_bpmn** enthält Informationen über die weiter analysierten BPMN-Prozesse im XML-Format. Das Attribut *path\_file* referenziert gleichnamiges Attribut aus der **result\_files** Tabelle und hat die gleiche Bedeutung. Das Attribut *valid* enthält Informationen über die Validierung von BPMN-Prozessen gegenüber der Sprachspezifikation (vergleiche folgenden Abschnitt 3.2.2). Im Attribut *constraints\_list* wird eine Liste von Regelverletzungen für einen Prozess gespeichert, die bei der Validierung eventuell gefunden wurden.

Die Tabelle **copy\_result\_files** enthält zwei Attribute: *path\_file* und *copy\_path\_file*. Das erste Attribut referenziert gleichnamiges Attribut aus der **result\_files** Tabelle und hat die gleiche Bedeutung. In dem zweiten Attribut wird der Pfad gespeichert, wo sich die Kopie der Datei befindet.

Die Tabelle **result\_projects** enthält Informationen über Repositorys, in denen mindestens ein potentieller BPMN-Prozess gefunden wurde. Zu diesen Informationen zählen: Zugangsdaten des Inhabers (*login*), Name des GitHub-Repository (*name*), die dominante Programmiersprache des Repository (*language*), Zeiten, zu denen das Repository erstellt wurde (*created\_at*) und das letzte Mal geändert wurde (*last\_commit\_at*). Im Attribut (*location\_country*) werden Namen der Lander gespeichert, aus denen die Beiträger des Repository stammen.



### 3.2.2. Das Mining-Programm

Das Mining-Programm, das mit Programmiersprache Python programmiert wurde, kann auf GitHub unter folgendem Link: [https://github.com/ViktorStefanko/BPMN\\_Crawler/](https://github.com/ViktorStefanko/BPMN_Crawler/) eingesehen werden [7].

Die Abbildung 3.4 zeigt uns den gesamten Prozess, der durch das Mining-Programm realisiert wird. Er besteht aus drei Schritten:

- (1) Im ersten Schritt werden potentielle BPMN-Prozesse aus öffentlichen GitHub-Repositorys gesammelt.
- (2) Im zweiten Schritt wird aus der Menge der potentiellen BPMN-Prozesse eine Untermenge der BPMN-Prozesse im XML-Format bestimmt.
- (3) Im letzten Schritt werden statistische Informationen über BPMN-Prozesse im XML-Format, potentielle BPMN-Prozesse und Repositorys, wo sie sich befinden, ermittelt und in der Datenbank gespeichert.

Der oben genannte erste Schritt entspricht dem Datenabruf und zweiter und dritter Schritte implementieren die Datenextraktion von allgemeinem Algorithmus für das MSR, siehe Abbildung 2.4.

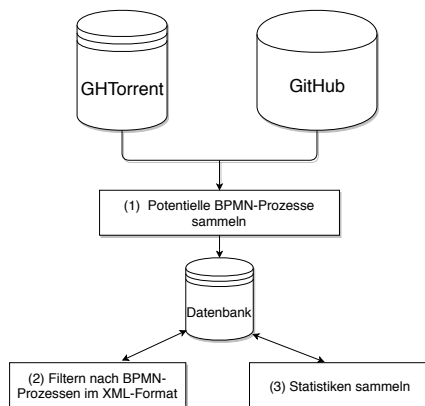


Abbildung 3.4.: Prozessdiagramm des Mining-Programms

Das Klassendiagramm 3.5 zeigt die Struktur des Mining-Programms. Es sind neun Klassen zu sehen. Für die jeweiligen Klassen sind die Funktionen mit ihren Rückgabetypen abgebildet. Funktionsparameter und Attribute wurden sowohl aus Platzspargründen, als auch, weil sie in diesem Kontext nicht von besonderer Bedeutung sind, weggelassen. Die Funktionen der Klassen *RepositoryCrawler* und *TreeCrawler* haben wir von Gregorio Robles entwickeltem „2016-uml-miner“-Programm [36], [15], [26], [38], [37] übernommen und für unsere Zwecke angepasst.

Die **GitHubApiCrawler** Klasse enthält die Funktion `run_api_crawler()`, die vollautomatisiert den ersten Schritt aus Abbildung 3.4 implementiert. Sie bekommt als Parameter zwei Integer Variablen, die eine untere und eine obere Schranke für das Attribut `id` von Repositorys in der `to_query_projects` Datenbanktabelle definiert, zwischen denen alle Repositorys nach BPMN-Prozessen zu durchsuchen sind. Die Schranken dienen dazu, das Mining-Programm parallel ausführen zu können, so dass jede Programminstanz eine Teilmenge aller zu durchsuchenden Repositorys

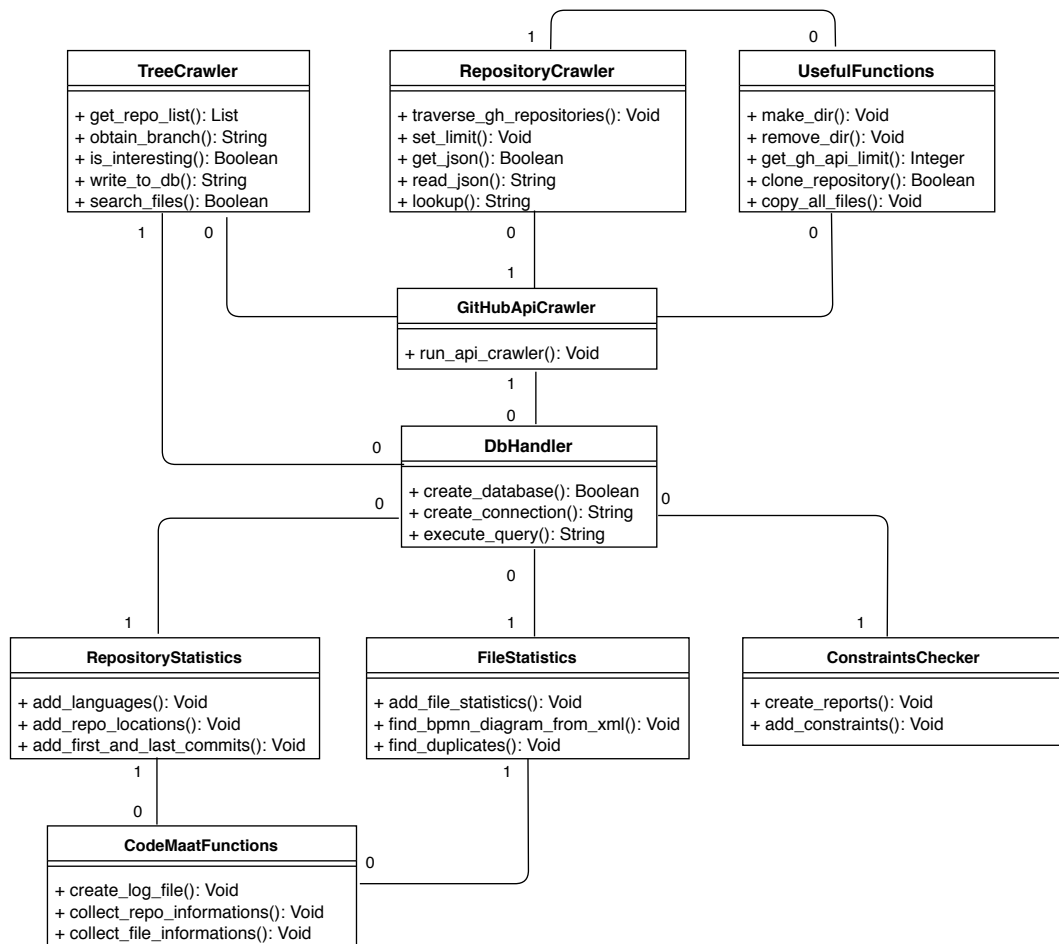


Abbildung 3.5.: Klassendiagramm des Mining-Programms

zugewiesen bekommt. Dann wird eine Liste der Namen und Zugangsdaten von Repositories aus der *to\_query\_projects* Datenbanktabelle ermittelt, die noch nicht durchsucht wurden. Für eine bestimmte Teilmenge aus dieser Liste werden die GitHub-Repositories nach BPMN-Prozessen durchsucht. Danach wird das *status* Attribut in der *to\_query\_projects* Tabelle auf 1 gesetzt, was bedeutet, dass diese Repositories schon durchsucht wurden. Schließlich wird dieser Vorgang für die nächste Teilmenge der Liste wiederholt. Die Funktion `run_api_crawler()` terminiert, wenn die Liste komplett bearbeitet wurde. Für die Erledigung der Einzelaufgaben benutzt `run_api_crawler()` Funktionen aus den **RepositoryCrawler**, **TreeCrawler** und **UsefulFunctions** Klassen.

Die **RepositoryCrawler** Klasse hat die zentrale Bedeutung im Programm. Wie im Abschnitt 3.1.3 beschrieben wurde, definiert die GitHub API 5000 Anfragen pro Stunde und Konto für authentifizierte Benutzer. Um diese Anzahl nicht zu überschreiten, prüft die Funktion `set_limit()` die Anzahl der verfügbaren GitHub API Anfragen und wenn sie verbraucht wurden, pausiert das Programm, bis neue 5000 Anfragen nach einer Stunde wieder zur Verfügung stehen. Andere Funktionen dieser Klasse (`traverse_gh_repositories()`, `get_json()`, `read_json()`, `lookup()`) erledigen den im Abschnitt 3.1.3.2 beschriebenen Vorgang, das heißt, für jedes GitHub-Repository wird seine Ordner- und Dateistruktur mithilfe der API ermittelt und in einer JSON-Datei gespeichert.

Die **TreeCrawler** Klasse enthält Funktionen, die es möglich machen, potentielle BPMN-Prozesse innerhalb des jeweiligen Repository zu finden. Die *get\_repo\_list()* Funktion ermittelt aus einem Ordner mit heruntergeladenen JSON-Dateien die Namen und Inhaber von Repositories und liefert sie in einer Liste zurück. Dann wird die *search\_files()* Funktion aufgerufen, wobei für jedes Repository aus der Liste seine Baumstruktur traversiert wird und für jeden Dateinamen die *is\_interesting()* Funktion aufgerufen wird. Letztere überprüft, ob sich in dem als Parameter übergebenen Dateinamen einschließlich der Dateiendung das gesuchte Schlüsselwort **bpmn** befindet und liefert im Erfolgsfall *Wahr* oder sonst *Falsch* zurück. Falls das Schlüsselwort **bpmn** ein Teil des Dateinamens ist, wird die Funktion *obtain\_branch()* aufgerufen, um *branch*, indem die Datei sich befindet, zu ermitteln. Dies wird benötigt, um das Weblink für den gefundenen potentiellen BPMN-Prozess zu bilden. Nachdem das passiert ist, wird Funktion *write\_to\_db()* aufgerufen. In dieser werden die ermittelten Daten, das heißt Name des Repository, Zugangsdaten, Weblink und Dateipfad in der Datenbanktabelle *result\_files* gespeichert.

Es ist wichtig, nachdem ein potentieller BPMN-Prozess gefunden wurde, diesen für die spätere Analyse zu sichern. Denn der Inhaber der Datei kann sie umbenennen, verschieben oder löschen, aufgrund dessen sie unter dem in der *result\_files* Datenbanktabelle gespeichertem Link nicht mehr verfügbar wird. Dafür wird in der *search\_files()* Funktion für jedes GitHub-Repository, in dem mindestens ein potentieller Prozess gefunden wurde, die *clone\_repository()* Funktion aus der **UsefulFunctions** Klasse aufgerufen und das Repository mit dem Befehl *git clone* heruntergeladen.

Die Funktionen der **UsefulFunctions** Klasse dienen Zwecken, wie beispielsweise Erzeugen und Löschen eines Ordners oder Kopieren von Dateien. Die Funktionen werden aus anderen Klassen aufgerufen.

Die Klasse **DbHandler** spielt die Rolle des Vermittlers zwischen dem Programm und der Datenbank. Sie enthält Funktionen für das Erzeugen einer Datenbank: *create\_database()*, für das Herstellen einer Verbindung zu einer Datenbank: *create\_connection()* und Funktion für die Ausführung von Datenbankabfragen: *execute\_query()*.

In der *execute\_query()* Funktion wurde der Fall berücksichtigt, wenn mehrere Instanzen des Programms gleichzeitig versuchen in die selbe Tabelle zu schreiben. In diesem Fall wird nur ein Schreibvorgang zu gleichen Zeit durchgeführt. Versuchen andere Programminstanzen parallel einen Schreibvorgang, wird eine Ausnahme geworfen. Wenn das passiert, wird das betroffene Programm für 0.3 sec pausiert und danach ein neuer Versuch für den Schreibvorgang unternommen. Das wird wiederholt, bis der Schreibvorgang erfolgreich durchgeführt werden kann.

Nachdem alle Repositories aus der Datenbanktabelle **to\_query\_projects** durchsucht worden sind, befinden sich Informationen über gesammelte potentielle BPMN-Prozesse in der **result\_files** Tabelle. Der nächste, zweite, Schritt ist somit, wie in Abbildung 3.4 zu sehen ist, die Ergebnisse nach BPMN-Prozessen im XML-Format zu filtern. Diese Aufgabe übernimmt die *find\_bpmn\_diagram\_from\_xml()*

Funktion aus der **FileStatistics** Klasse. Es wird zuerst eine Liste von Textdateien gebildet, die im XML-Format gespeichert sind. Danach wird in jeder Datei nach der „<http://www.omg.org/spec/BPMN/20100524/MODEL>“ Sequenz gesucht. Denn sie soll nach Spezifikation von BPMN [10], als Wert des `xmlns` Attributs im einleitenden Tag stehen. Für jede XML-Datei, die diese Sequenz enthält, wird ihr Dateipfad in die **result\_bpmn** Datenbanktabelle gespeichert.

Nach der Filterung beginnt somit der letzte, dritte, Schritt aus der Abbildung 3.4. Hier werden verschiedene Informationen sowohl über BPMN-Prozesse, als auch über Repositories, zu denen sie gehören, gesammelt. Dafür wurden im Mining-Programm drei externe Werkzeuge benutzt:

- Code Maat [12] - ein Programm, mit dem Daten aus Versionskontrollsystemen abgerufen und analysiert werden können [42]. Code Maat benutzt log-Datei mit der Änderungshistorie eines Repository, um solche Metriken, wie Anzahl der Commits und Autoren oder Dateialter zu berechnen.
- Duplicate Files Finder [14] - eine Anwendung, die nach doppelten Dateien sucht (Dateien mit demselben Inhalt, aber nicht unbedingt demselben Namen) und dem Benutzer das Entfernen doppelter Dateien ermöglicht, indem sie entweder gelöscht oder Verknüpfungen erstellt werden“ [14]. Bei der Suche nach Duplikaten werden zuerst alle Dateien nach ihrer Größe sortiert und dann Dateien mit gleicher Größe miteinander verglichen [14].
- BPMNspector [9] - ein Programm, das einzelne Dateien oder vollständige Verzeichnisse auf BPMN-Dateien überprüft und Verstöße gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln meldet [9].

Die Klasse **CodeMaatFunctions** enthält drei Funktionen: *create\_log\_file()*, *collect\_file\_informations()* und *collect\_repo\_informations()*. Die *create\_log\_file()* Funktion ist dafür zuständig, für ein GitHub Repository die gesamte Historie seiner Commits zu bilden und in einer log-Datei zu speichern. In den *collect\_file\_informations()* und *collect\_repo\_informations()* Funktionen wird das Code Maat aufgerufen und log-Dateien als Eingabequelle benutzt.

Die *collect\_file\_informations()* Funktion erstellt für ein geklontes GitHub-Repository zwei csv-Dateien. In erster werden für jede Datei innerhalb des jeweiligen Repository die Anzahl der Autoren und Dateiänderungen gespeichert. In der zweiten Datei werden für jede Datei innerhalb des jeweiligen Repository ihr Alter in Monaten (Zeitraum seit Dateierstellung) gespeichert.

Die *collect\_repo\_informations()* Funktion erzeugt für ein GitHub-Repository eine csv-Datei, in der steht, an welchem Tag wie viel Dateien erzeugt und gelöscht wurden und wie viel Commits gemacht wurden.

Die **FileStatistics** Klasse enthält außer der schon beschriebenen Funktion *find\_bpmn\_diagram\_from\_xml()* auch noch die Funktionen *add\_file\_statistics()* und *find\_duplicates()*. Die *add\_file\_statistics()* Funktion benutzt *create\_log\_file()* und *collect\_file\_informations()* als Unterroutrinen und speichert in der *file\_statistics* Datenbanktabelle für jede Datei, die sich dort befindet, das Dateialter in Monaten und

die Anzahl der Autoren und Änderungen.

Vor der Benutzung der *find\_duplicates()* Funktion soll die *copy\_all\_files* Funktion aus **UsefulFunctions** Klasse aufgerufen werden. Die letzte kopiert alle Dateien aus den geklonten GitHub-Repositorys in einen Ordner, dessen Name in der *result\_files* Datenbanktabelle stehen. Die Dateikopien werden dabei umbenannt, denn es kann sein, dass manche Dateien aus unterschiedlichen Repositorys den gleichen Namen haben können. Der Pfad der Datei und der Name ihrer Kopie werden in der **copy\_result\_files** Datenbanktabelle gespeichert.

Danach kann die *find\_duplicates()* Funktion aufgerufen werden, die das Programm Duplicate Files Finder als Unteroutine nutzt. Duplicate Files Finder erstellt aus dem Ordner mit Dateikopien eine txt-Datei mit Dateinamen, die Duplikate haben. Diese Datei wird dann weiter in *find\_duplicates()* Funktion analysiert und für jede zwei und mehr Dateien, deren Inhalte gleich sind, wird die gleiche Integer Zahl in der *duplicate* Spalte von **result\_files** Datenbanktabelle gespeichert.

Die **RepositoryStatistics** Klasse enthält Funktionen, deren Aufgabe es ist, Informationen über geklontes GitHub-Repository zu gewinnen und in die **result\_projects** Datenbanktabelle zu schreiben. Die *add\_languages()* Funktion speichert für jedes GitHub-Repository die dominierende Programmiersprache, die mittels GitHub API abgefragt wird.

Die *add\_repo\_location()* Funktion berechnet für jedes GitHub-Repository Ländercodes für die Länder, in denen sich die am Repository Mitwirkenden befinden, falls die Information vorliegt. Das wird in zwei Schritten gemacht. In erstem Schritt wird für jedes Repository die Liste der Mitwirkenden ermittelt. Das wird mithilfe von HTTP-Anfragen an die GitHub API gemacht. Im zweiten Schritt wird für jeden Entwickler aus der Liste sein Standort ermittelt, indem diese Information aus der Datenbanktabelle **users** entnommen und in der Datenbanktabelle **result\_projects** gespeichert wird.

Die *add\_first\_and\_last\_commits()* Funktion benutzt als Unteroutine die *collect\_repo\_informations()* Funktion aus der **CodeMaatFunctions** Klasse und berechnet das erste und das letzte Commit, das ein geklontes GitHub-Repository hat. Diese Information wird in der **result\_projects** Datenbanktabelle gespeichert.

Die **ConstraintsChecker** Klasse wird verwendet, um die im XML-Format vorliegenden BPMN-Prozesse zu analysieren. Die *create\_reports()* Funktion benutzt als Unteroutine **BPMNSpector**. Dieses erzeugt für jeden BPMN-Prozess einen Bericht in XML-Format, in dem steht, ob der Prozess Verstöße gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln hat und wenn dass der Fall ist, werden sie aufgelistet.

Als Beispiel für einen Regelverstoß gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln betrachten wir einen der häufigsten Fehler in BPMN-Prozessen: eine Verletzung einer `<sequenceFlow>` Einschränkung (in **BPMNSpector** mit „EXT.023“ bezeichnet) [8]. Die Abbildung 3.6 zeigt uns einen einfachen BPMN-Prozess, der weiter in den Abbildungen 3.7 und 3.8 im XML-Format

zu sehen ist. Der in der Abbildung 3.7 dargestellte Prozess verletzt die <sequence-Flow> Einschränkung, indem in den Definitionen vom Startereignis, Endereignis und Task die ein- oder ausgehende Kante (Sequence Flow) nicht erwähnt wird. Die korrekte Version vom Prozess 3.7 ist in der Abbildung 3.8 zu sehen.



Abbildung 3.6.: Ein einfacher BPMN-Prozess

```

1 <process id="P1" name="A Basic Process" isExecutable="true">
2   <startEvent id="start" name="Start"></startEvent>
3
4   <sequenceFlow id="flow1" sourceRef="start" targetRef="task"></sequenceFlow>
5
6   <task id="task" name="Do The Work!"></task>
7
8   <sequenceFlow id="flow2" sourceRef="task" targetRef="end"></sequenceFlow>
9
10  <endEvent id="end" name="The End."></endEvent>
11 </process>
  
```

Abbildung 3.7.: Ein einfacher BPMN-Prozess mit Verletzung einer <sequence-Flow> Einschränkung. Übernommen aus [8].

```

1 <process id="P1" name="A Basic Process" isExecutable="true">
2   <startEvent id="start" name="Start">
3     <outgoing>flow1</outgoing>
4   </startEvent>
5
6   <sequenceFlow id="flow1" sourceRef="start" targetRef="task"></sequenceFlow>
7
8   <task id="task" name="Do The Work!">
9     <incoming>flow1</incoming>
10    <outgoing>flow2</outgoing>
11  </task>
12
13  <sequenceFlow id="flow2" sourceRef="task" targetRef="end"></sequenceFlow>
14
15  <endEvent id="end" name="The End.">
16    <incoming>flow2</incoming>
17  </endEvent>
18 </process>
  
```

Abbildung 3.8.: Ein einfacher BPMN-Prozess ohne Verletzung einer <sequence-Flow> Einschränkung. Übernommen aus [8].

In der `add_constraints()` Funktion werden die erzeugte Berichte analysiert. Für jeden Prozess wird geprüft ob er valide ist und wenn ja, das *valide* Attribut in der **result\_bpmn** Datenbanktabelle auf 1 gesetzt, wenn nein - auf 0. Falls der Prozess nicht valide ist, d.h. BPMNspector einige Verstöße gefunden hat, werden die Namen der Verstöße und ihre Anzahl innerhalb des Prozesses zu einer Liste gebildet und in dem *constraints\_list* Attribut in der **result\_bpmn** Datenbanktabelle gespeichert.

### 3.3. Ergebnisse

Es wurde der GHTorrent-Dump vom 1.11.2018 genutzt, in dem es 61.632.173 nicht-gelöchte und nicht-geforkte Repositories gibt. Damit der Mining-Prozess nicht länger als 1,5 Monate dauert, wurde die Entscheidung getroffen, nur 10 Prozent, also

6.163.217 Repositorys zufällig auszuwählen und nach BPMN-Prozessen zu durchsuchen.

Das implementierte Mining-Programm wurde dazu auf 4 Rechnern im Linux-Pool 1 der Fakultät für Mathematik und Informatik an der Friedrich Schiller Universität Jena ausgeführt. Jedes Programm hatte die Aufgabe, circa 1.540.804 Repositorys zu durchsuchen.

Die Arbeit des Programms hat auf einem der 4 Rechner durchschnittlich 30 Tage, 12 Stunden und 48 Minuten gedauert. Die aufsummierte Laufzeit von 4 Rechnern beträgt somit 122 Tage, 3 Stunden und 11 Minuten.

Diese Daten lassen zu, die mittlere Anzahl von Repositorys, die pro Stunde auf einem der 4 Rechner durchsucht wurden, zu berechnen. Es wurden also pro Stunde circa 2.103 Repositorys durchsucht.

Es muss erwähnt werden, dass es während der Laufzeit der Programme einige Unterbrechungen gab. Dies passierte, wenn einer der 4 Rechner ausgeschaltet wurde. Die Programme wurden jedoch mittels SSH-Verbindung periodisch manuell geprüft, und falls nötig neu gestartet, so dass Unterbrechungen nicht länger als 3 Stunden andauerten.

Es wurden 1.251 Repositorys gefunden (von 6.163.217 durchsuchten Repositorys), die mindestens einen potentiellen BPMN-Prozess (es kommt „bpmn“ im Dateinamen vor) enthielten. Insgesamt wurden 21.306 potentielle BPMN-Prozesse gefunden.

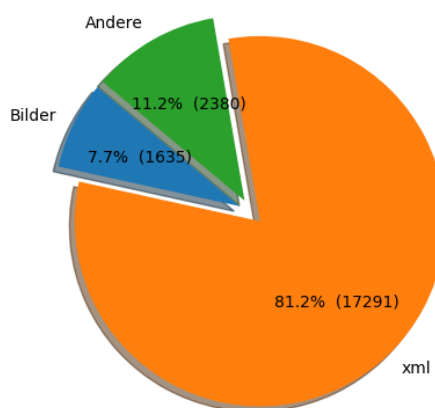


Abbildung 3.9.: Dateiformate von potentiellen BPMN-Prozessen (gruppiert)

Das Diagramm 3.9 zeigt eine Übersicht zu den dabei gefundenen Dateiformaten. Es ist zu sehen, dass die absolute Mehrheit (81,2%) der gefundenen Dateien das XML Format haben. Ihre Anzahl ist 17.291. Der Anteil an Bilddateien und Dokumenten, zu denen Dateien mit den Dateierweiterungen \*.png, \*.gif, \*.jpeg, \*.jpg, \*.svg und \*.pdf zählen, beträgt 7,7% oder 1.635 Dateien. Der Anteil aller anderen Dateiformate, beispielsweise Dateien mit den Dateierweiterungen \*.js oder \*.jar, beträgt 11,2% von allen gefundenen Dateien.

Die 17.291 Dateien im XML-Format wurden weiter analysiert. Als Ergebnis der Analyse, vergleiche auch den zweiten Schritt in Abbildung 3.4, ergab sich eine weitere Reduktion auf 16.907 XML-Dateien, die als BPMN-Prozesse erkannt wurden. Diese Dateien stammten aus 928 GitHub-Repositorys.

In der Tabelle 3.1 sind Resultate des Mining-Programms dargestellt.

	Dateien	Repositorys
Insgesamt	21.306	1.251
BPMN-Prozesse im XML-Format	16.907	928

Tabelle 3.1.: Resultate des Mining-Programms



## 4. Analyse gefundener BPMN-Dateien

Dieses Kapitel enthält Ergebnisse der Analysen für die Daten, die durch das im vorangegangenen Abschnitt beschriebene Mining-Programm gesammelt wurden. Hier werden zuerst allgemeine Statistiken über gefundene Dateien und ihre Repositories dargestellt und anschließend Statistiken über die Korrektheit der BPMN-Prozesse im XML-Format, die mit Hilfe des Analysewerkzeugs BPMNspector generiert wurden, präsentiert. Dieses Kapitel bildet somit den letzten Schritt aus dem allgemeinen Algorithmus für das MSR, die Datenanalyse, ab (siehe Abb. 2.4).

### 4.1. Allgemeine Statistiken

Wir haben die allgemeinen Statistiken in zwei Gruppen unterteilt: Statistiken über GitHub-Repositories, die BPMN-Prozesse enthalten, und Statistiken über die gefundenen BPMN-Dateien selbst.

#### 4.1.1. Statistiken über GitHub-Repositories

Die Repositories mit BPMN-Prozessen wurden hinsichtlich der dominanten Programmiersprache, dem Zeitpunkt der Erstellung (Alter), dem Zeitpunkt der letzten Änderung und dem Standort der Repositoryentwickler untersucht.

##### 4.1.1.1. Dominante Programmiersprache

Diese Statistik wurde mittels GitHub API ermittelt, siehe Funktion `add_languages()` im Mining-Programm [34] und Beschreibung im Abschnitt 3.2.2.

Die Abbildung 4.1 zeigt die dominanten Programmiersprachen in den GitHub-Repositories, die BPMN-Prozesse enthalten. Aus diesem Diagramm kann man ablesen, dass jedes zweite Repository (51,2%) Java als dominante Programmiersprache hat und mehr als jedes dritte Repository (37%) - JavaScript. Diesen Sprachen mit einem großen Abstand folgen die Beschreibungssprache HTML (7,1%) und andere Programmiersprachen (<2%).

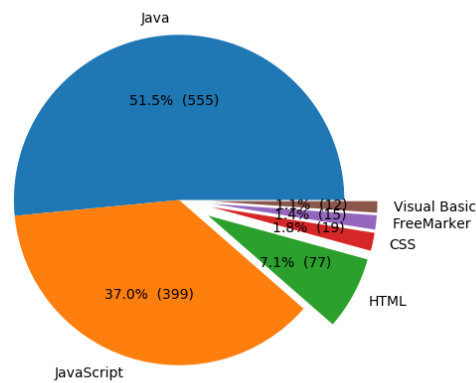


Abbildung 4.1.: Dominante Programmiersprachen der Repositorys mit BPMN-Dateien

Um festzustellen, ob es Unterschiede zwischen den oben genannten Statistiken über dominante Programmiersprachen von GitHub-Repositorys mit BPMN-Prozessen und allen GitHub-Repositorys gibt, schauen wir jetzt auf die Abbildung 4.2. Diese zeigt die dominanten Programmiersprachen von allen öffentlichen GitHub-Repositorys, erstellt im 1. Quartal 2018. Es ist zu sehen, dass JavaScript, Python und Java zu den dominanten Programmiersprachen auf GitHub gehören. Der Vergleich der Abbildungen 4.1 und 4.2 zeigt uns, dass es einige Ähnlichkeiten und Unterschieden zwischen ihnen gibt. So gehören in beiden Abbildungen JavaScript und Java zu den zwei meistgenutzten Programmiersprachen. Jedoch bei Repositorys mit BPMN-Prozessen ist Java an erster Stelle und bei allen GitHub-Repositorys - JavaScript. Auch alle anderen Beschreibung- und Programmiersprachen aus der Abbildung 4.1, wie HTML, CSS, FreeMarker und Visual Basic gehören allgemein nicht zu den am häufigsten auftretenden Programmiersprachen auf GitHub.

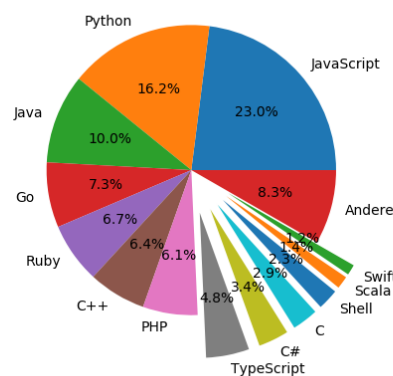


Abbildung 4.2.: Dominante Programmiersprachen von allen öffentlichen GitHub-Repositorys, erstellt im 1. Quartal 2018 [21]

#### 4.1.1.2. Erstellungsjahre und Jahre der letzten Änderung

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_commits()` im Mining-Programm [34] und Beschreibung im Abschnitt 3.2.2.

Das Kreisdiagramm in der Abbildung 4.3 zeigt uns eine Dynamik der Erstellung von neuen GitHub-Repositorys mit BPMN-Prozessen. Es ist zu sehen, dass es zwischen 2005 und 2010 genau so viele Repositorys erstellt wurden, wie in 2011. Nach 2011 wurde diese Anzahl jedes Jahr deutlich höher, als im Jahr davor. Der Grund dafür besteht vermutlich darin, dass im Januar 2011 die neue Version 2.0 von BPMN erschien. Seitdem wuchs die Anzahl der neu-erstellten Repositorys stetig und zum Beispiel wurden in 2017 fast 5-Mal so viele Repositorys mit BPMN-Prozessen erstellt, als drei Jahre davor in 2014. Im Durchschnitt betrug der Zuwachs der GitHub-Repositorys mit BPMN-Prozessen zwischen 2011 und 2017 ca. 71.7% pro Jahr. Die Daten für 2018 sind nicht komplett. Das liegt daran, dass wir den GHTorren-Datensatz von 01.11.2018 benutzt haben, der Informationen nur bis zu diesem Datum enthielt.

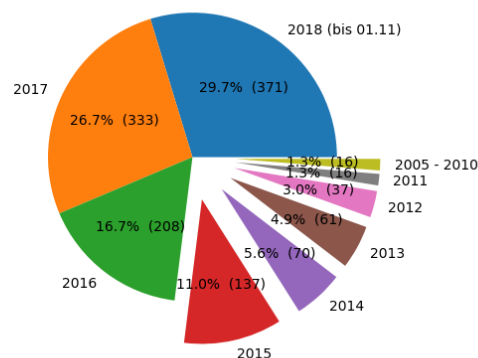


Abbildung 4.3.: Jahre, in denen Repositorys mit BPMN-Dateien erstellt wurden

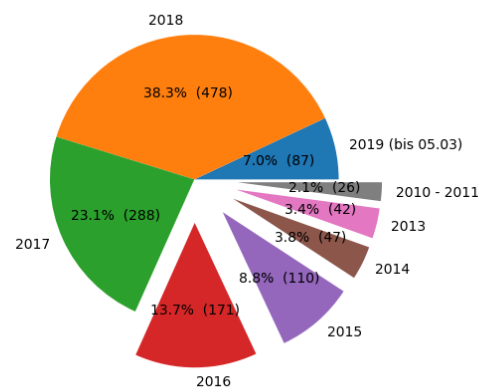


Abbildung 4.4.: Jahre, in denen Repositorys mit BPMN-Dateien letztes Mal geändert wurden

Das Diagramm in der Abbildung 4.4 liefert uns Informationen darüber, wann GitHub-Repositorys mit BPMN-Prozessen das letzte Mal geändert wurden. Dieses Diagramm zeigt, dass im Unterschied zur Abbildung 4.3, wo die ersten Repositorys im Jahre 2005 erstellt wurden, die ersten Repositorys im Jahre 2010 abgeschlossen wurden. D.h. alle zwischen 2005 und 2010 erstellten Repositorys waren mindestens bis 2010 aktiv. Aus dem Diagramm kann man auch entnehmen, dass die Anzahl

der zum letzten Mal geänderten Repositorys jedes Jahr zwischen 2010 und 2018 immer größer war als im Jahr davor. Die Daten für das Jahr 2019 sind nicht komplett - nur bis 5. März. Das liegt daran, dass das Klonen der GitHub-Repositorys am 05.03.2019 abgeschlossen wurde und wir Informationen über GitHub-Repositorys nur bis zu diesem Datum hatten.

Um die Abbildungen 4.3 und 4.4 zu vergleichen, wurde aus beiden Grafiken ein gemeinsames Diagramm 4.5 mit absoluter Häufigkeit erstellt und zwar für die Jahre zwischen 2010 und 2017, für die es komplette Informationen auf beiden Grafiken gab. Das Diagramm 4.5 zeigt uns deutlich, dass die Anzahl der neu-erstellten Repositorys mit BPMN-Prozessen in einem der betrachteten Jahren größer war, als die Anzahl der Repositorys, die in jeweiligem Jahr zum letzten Mal geändert wurden. Daraus kann man schließen, dass in diesen Jahren mehr öffentlichen GitHub-Repositorys mit BPMN-Prozessen erstellt, als abgeschlossen wurden.

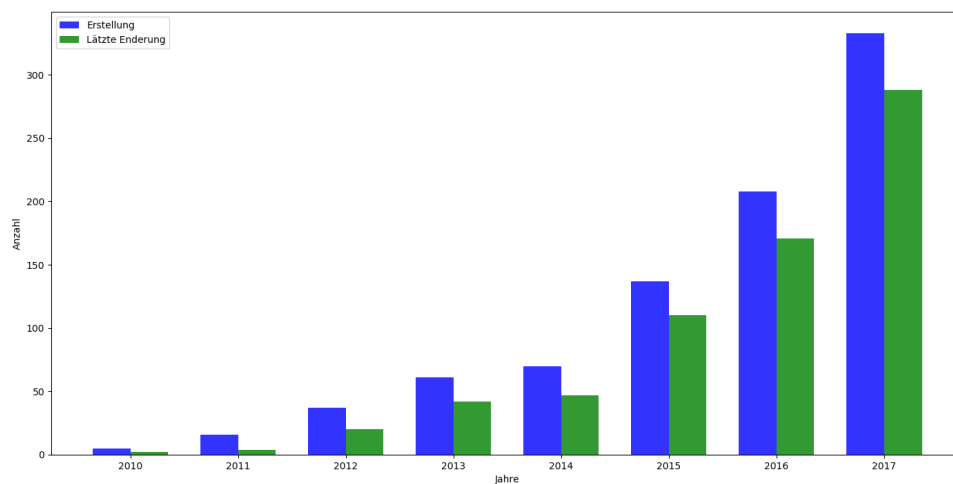


Abbildung 4.5.: Jahre, in denen Repositorys mit BPMN-Dateien erstellt und letztes Mal geändert wurden

#### 4.1.1.3. Dauer der Aktivität

Die Abbildungen 4.6 und 4.7 zeigen Informationen über Dauer der Aktivität (Zeitraum zwischen Erstellung und letzter Änderung) von Repositorys mit BPMN-Dateien. Es ist zu sehen, dass 29,1% (364) der Repositorys nur 1 Tag aktiv waren und 25,1% (314) der Repositorys - nur ein Monat. D.h. mehr als die Hälfte der Repositorys weniger als ein Monat aktiv waren. Fast 20% (248) der Repositorys waren von 1 bis 6 Monaten aktiv. Signifikant ist, dass es 48 Repositorys gibt, die mehr als 5 Jahre aktiv waren. 2 Repositorys haben Aktivität von 11 Jahren und ein Repository sogar von 12 Jahren.

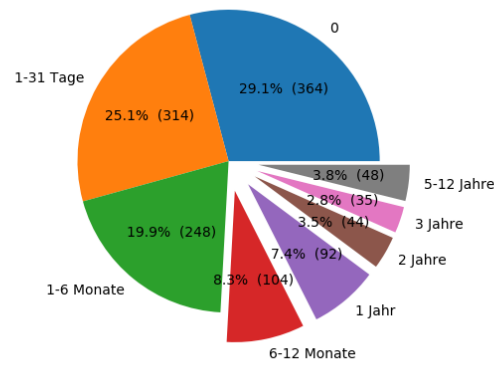


Abbildung 4.6.: Dauer der Aktivitäten von GitHub-Repositorys mit BPMN-Dateien (Kreisdigramm)

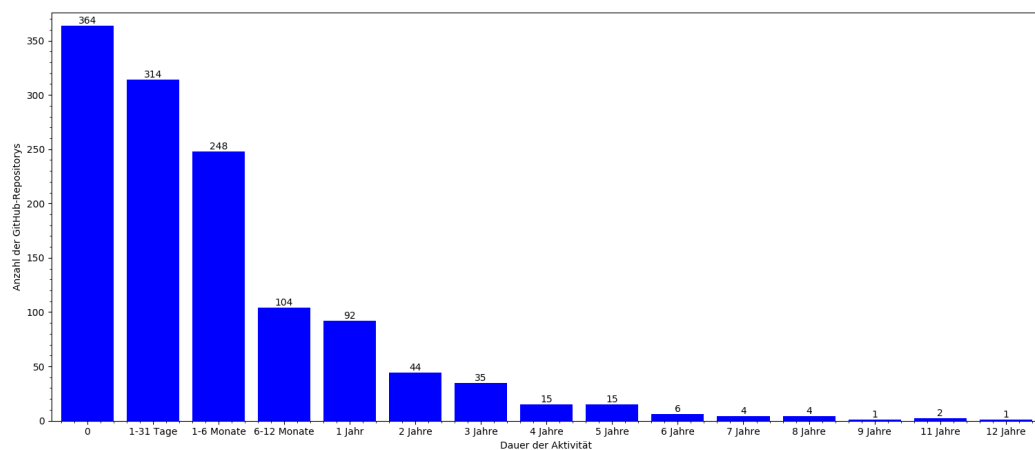


Abbildung 4.7.: Dauer der Aktivitäten von GitHub-Repositorys mit BPMN-Dateien (Balkendiagramm)

#### 4.1.1.4. Anzahl der Commits

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_commits()` im Mining-Programm [34] und Beschreibung im Abschnitt 3.2.2.

Das nächste Merkmal von Repositorys mit BPMN-Dateien, das untersucht wurde, war die Anzahl der Commits, die seit der Repositoryerstellung gemacht wurden. Aus der Abbildung 4.8 kann man entnehmen, dass 12,5% der Repositorys nur ein Commit hat und 24,1% von 2 bis 5 Commits. Insgesamt kann man aus diesem Diagramm entnehmen, dass mehr als die Hälfte der Repositorys 10 oder weniger Commits haben. Fast jedes vierte Repository hat zwischen 11 und 50 Commits. 5,5% der Repositorys haben mehr als 500 Commits. Die größte Anzahl der Commits, die ein Repository hat, beträgt 10056.

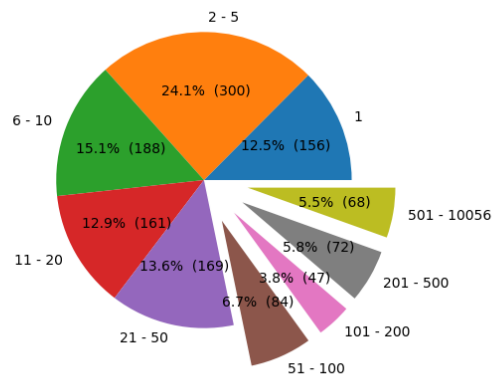


Abbildung 4.8.: Anzahl der Commits von GitHub-Repositorys mit BPMN-Dateien

#### 4.1.1.5. Standorte der Repository-Entwickler

Diese Statistik wurde mittels GitHub API und Daten des GHTorrent-Projekts ermittelt, siehe Funktion `add_repo_location()` im Mining-Programm [34] und Beschreibung im Abschnitt 3.2.2.

Ein weiteres Merkmal zu den Repositorys, das untersucht wurde, war der Standort der Repository-Entwickler. Die Ergebnisse sind in der Abbildung 4.9 dargestellt. Diese Grafik wurde jedoch nur anhand der Standorte von 395 GitHub-Repositorys mit BPMN-Prozessen gebildet. Das liegt daran, dass die Datenbanktabelle „users“ (Abbildung 3.3), deren Inhalt aus dem Datensatz des GHTorrent-Projekts stammt und die Informationen über alle GitHub-Benutzer enthält, für viele von ihnen im Feld „country\_code“ keine Angabe hat.

Bei Betrachten der Abbildung 4.9 fällt sofort auf, dass die meisten Entwickler, die zu den GitHub-Repositorys mit BPMN-Prozessen ihr Code schrieben, aus China (14,7%), Deutschland (14,4%) und aus den USA (14,4%) stammen. Ihnen mit einem großen Abstand folgen Programmierer aus der Schweiz (3,8%), Frankreich (3,5%) und anderen Ländern.

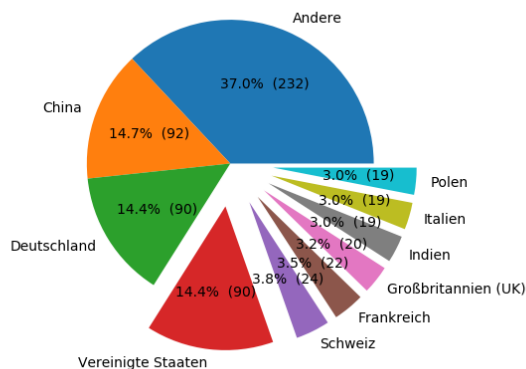


Abbildung 4.9.: Standorte der Repository-Entwickler

Die Abbildung 4.10 enthält die Länder, aus denen die meisten GitHub-Benutzer

kommen. Diese Abbildung ähnelt der Abbildung 4.9 insofern, dass in beiden USA und China die zwei Länder mit den meisten GitHub-Benutzern sind. Aber im Gegensatz zur Abbildung 4.9, wo Deutschland und USA den zweiten Platz teilen, befindet sich dieses europäische Land in der Abbildung 4.10 auf dem 5. Platz. Insgesamt kommen in beiden Abbildungen sechs dieselbe Länder vor.

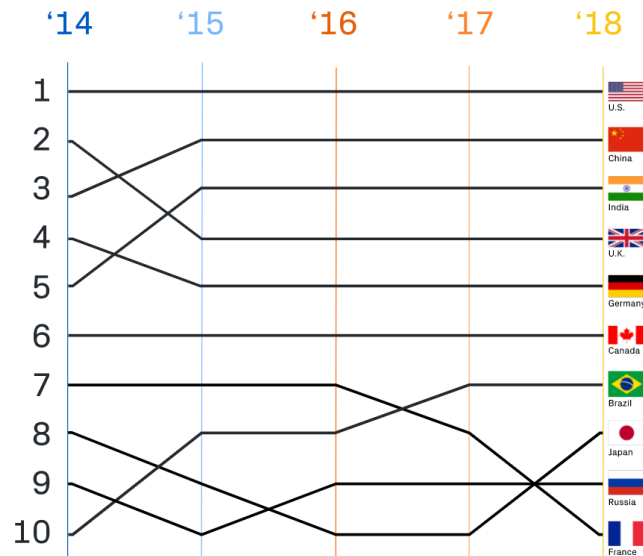


Abbildung 4.10.: Standorte mit den meisten GitHub-Benutzer [32]

#### 4.1.2. Statistiken über gefundene BPMN-Dateien

Die gefundenen BPMN-Prozesse, gespeichert im XML-Format und alle anderen Dateien, die in ihrem Namen das Wort „bpmn“ enthalten, wurden zusammen hinsichtlich der Anzahl von Duplikaten, Autoren, Dateialter und der Anzahl an Dateiänderungen analysiert. Diese beide Gruppen von Dateien wurden zusammen analysiert, da es bei der getrennten Betrachtung keine großen Unterschiede gab.

##### 4.1.2.1. Analyse der Dateien bezüglich Duplikate

Diese Statistik wurde mittels des Programms Duplicate Files Finder ermittelt, siehe Funktion `find_duplicates()` im Mining-Programm [16] und Beschreibung im Abschnitt 3.2.2.

In der Abbildung 4.17 sind Informationen zu Duplikaten von BPMN-Dateien dargestellt. Es ist zu sehen, dass nur für 37,7% der Dateien keine Duplikate gefunden wurden. Der Anteil der Dateien, die mindestens einmal dupliziert wurden, beträgt 12,6% und der Anteil der reinen Duplikate - 49,7%.

Die gefundenen Duplikate (Kopien von anderen Dateien), wurden bei allen weiteren Statistik-Analysen außer Statistik über die Anzahl der BPMN-Prozessen im XML-Format mit der grafischen Repräsentation, Anzahl der XML-Elemente der BPMN-Prozesse und Statistik über die Korrektheit der BPMN-Prozesse im XML-Format (siehe 4.2) mitgezählt.

Die am meisten duplizierte BPMN-Prozesse sind im Anhang A zu sehen.

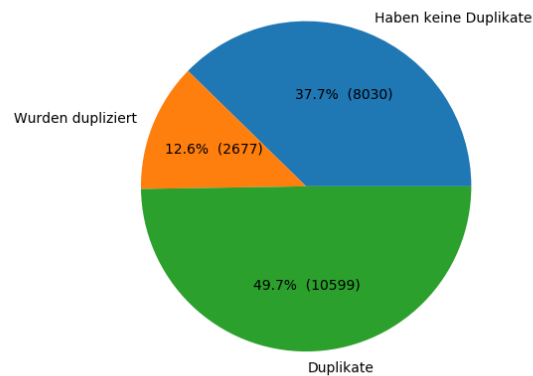


Abbildung 4.11.: Angaben über Duplikate von BPMN-Dateien

#### 4.1.2.2. Anzahl der BPMN-Prozessen im XML-Format mit der grafischen Repräsentation

Diese Statistik wurde mittels eines Python-Skripts ermittelt, siehe Funktion `count_process_elements()` im Mining-Programm [16] und Beschreibung im Abschnitt 3.2.2.

Die Abbildung 4.12 zeigt die XML-Repräsentation vom BPMN-Prozess 3.6 aus dem Abschnitt 3.2.2. An diesem Beispiel kann man sehen, dass ein BPMN-Prozess im XML-Format einen Wurzel-Element „definitions“ (in der Abb. 4.12 „bpmn:definitions“) und zwei Kinder-Elemente „process“ (in der Abb. 4.12 „bpmn:process“) und „BPMNDiagram“ (in der Abb. 4.12 „bpmn:BPMNDiagram“) hat. Das XML-Element „process“ definiert alle Elemente vom BPMN-Prozess und das XML-Element „BPMNDiagram“ enthält die grafische Anordnung der BPMN-Prozesselemente.



```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
3                   xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
4                   xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
5                   xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
6                   id="Definitions_1ygvvxc" targetNamespace="http://bpmn.io/schema/bpmn"
7                   exporter="Camunda Modeler" exporterVersion="2.0.3">
8   <bpmn:process id="Process_1" name="A Basic Process" isExecutable="true">
9     <bpmn:startEvent id="StartEvent_1" name="Start"></bpmn:startEvent>
10    <bpmn:task id="Task_1ve7ppt" name="Do The Work!"></bpmn:task>
11    <bpmn:sequenceFlow id="SequenceFlow_0kubrj1" sourceRef="StartEvent_1" targetRef="Task_1ve7ppt" />
12    <bpmn:endEvent id="EndEvent_0d529fy" name="The End."></bpmn:endEvent>
13    <bpmn:sequenceFlow id="SequenceFlow_1parnc1" sourceRef="Task_1ve7ppt" targetRef="EndEvent_0d529fy" />
14  </bpmn:process>
15  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
16    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Process_1">
17      <bpmndi:BPMNShape id="BPMNShape_StartEvent_2" bpmnElement="StartEvent_1">
18        <dc:Bounds x="173" y="102" width="36" height="36" />
19      <bpmndi:BPMNLabel>
20        <dc:Bounds x="178" y="146" width="25" height="14" />
21      </bpmndi:BPMNLabel>
22    </bpmndi:BPMNShape>
23    <bpmndi:BPMNShape id="Task_1ve7ppt_di" bpmnElement="Task_1ve7ppt">
24      <dc:Bounds x="278" y="80" width="100" height="80" />
25    </bpmndi:BPMNShape>
26    <bpmndi:BPMNEdge id="SequenceFlow_0kubrj1_di" bpmnElement="SequenceFlow_0kubrj1">
27      <di:waypoint x="209" y="120" />
28      <di:waypoint x="278" y="120" />
29    </bpmndi:BPMNEdge>
30    <bpmndi:BPMNShape id="EndEvent_0d529fy_di" bpmnElement="EndEvent_0d529fy">
31      <dc:Bounds x="477" y="102" width="36" height="36" />
32    <bpmndi:BPMNLabel>
33      <dc:Bounds x="472" y="144" width="45" height="14" />
34    </bpmndi:BPMNLabel>
35    </bpmndi:BPMNShape>
36    <bpmndi:BPMNEdge id="SequenceFlow_1parnc1_di" bpmnElement="SequenceFlow_1parnc1">
37      <di:waypoint x="378" y="120" />
38      <di:waypoint x="477" y="120" />
39    </bpmndi:BPMNEdge>
40  </bpmndi:BPMNPlane>
41 </bpmndi:BPMNDiagram>
42 </bpmn:definitions>

```

Abbildung 4.12.: Beispiel eines BPMN-Prozess im XML-Format

Da das XML-Element „BPMNDiagram“ nur die grafische Repräsentation eines Prozesses enthält, wird es oft in BPMN-Prozessen im XML-Format weggelassen. Die Abbildung 4.13 zeigt uns Statistik über das XML-Element „BPMNDiagram“ der gefundenen BPMN-Prozessen im XML-Format. In die Analyse wurden 8.904 Dateien von den insgesamt 16.907 gefundenen BPMN-Dateien im XML-Format einbezogen. Bei den restlichen 8.003 Dateien handelt es sich um Kopien dieser Dateien. Es ist zu sehen, dass nur 63,8% (5.679) der Dateien das XML-Element „BPMNDiagram“ haben und bei 35% (3.118) der Dateien dieses Element fehlt. Bei 107 Dateien konnte nicht ermittelt werden, ob sie das XML-Element „BPMNDiagram“ enthalten.

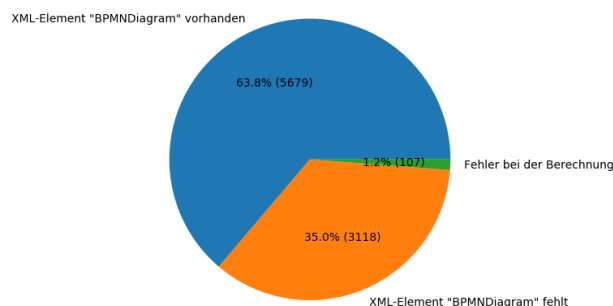


Abbildung 4.13.: Anzahl der BPMN-Prozesse im XML-Format mit dem XML-Element „BPMNDiagram“

#### 4.1.2.3. Anzahl der XML-Elemente der BPMN-Prozesse

Diese Statistik wurde mittels eines Python-Skripts ermittelt, siehe Funktion `count_process_elements()` im Mining-Programm [16] und Beschreibung im Abschnitt 3.2.2.

Ein weiteres interessantes Merkmal, was untersucht wurde, war die Berechnung der Größe von BPMN-Prozessen im XML-Format. Als Maß für die Größe eines Prozesses wurde die Anzahl der XML-Elemente vom XML-Element „process“ genommen. Das XML-Element „BPMNDiagram“ wurde bei der Berechnung ignoriert, weil es in 35% der Dateien im XML-Format nicht vorhanden ist.

In die Analyse wurden 8.904 BPMN-Prozesse von den insgesamt 16.907 gefundenen BPMN-Prozessen im XML-Format einbezogen. Bei den restlichen 8.003 BPMN-Prozessen handelt es sich um Kopien dieser Dateien. Für die 8.798 Dateien konnte die Anzahl der XML-Elemente berechnet werden und für die restlichen 107 Dateien, wegen einigen Ausnahmen bei der Berechnung, konnte es nicht gemacht werden.

Die Abbildung 4.14 zeigt die Ergebnisse dieser Analyse. Aus dem Diagramm kann man entnehmen, dass 21,1% (1.881) der BPMN-Prozesse von 1 bis 10 XML-Elemente und 30,5% (2.714) der BPMN-Prozesse zwischen 11 und 20 XML-Elemente haben. Folglich haben fast die Hälfte der BPMN-Prozesse (48,4%) mehr als 20 XML-Elemente. Interessant ist, dass 0,6% (57) der BPMN-Prozesse mehr als 1001 XML-Elemente haben. Die größte Anzahl der XML-Elemente, die ein gefundener BPMN-Prozess hat, beträgt 4.096.

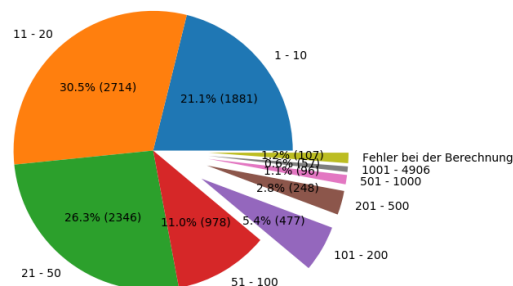


Abbildung 4.14.: Anzahl der XML-Elemente von BPMN-Dateien im XML-Format

Die 5 größten BPMN-Prozesse im XML-Format, die das XML-Element „BPMNDiagram“ haben und somit grafisch darstellbar sind, sind im Anhang B zu sehen.

#### 4.1.2.4. Anzahl der Autoren

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_age_n_authors_n_revers()` im Mining-Programm [16] und Beschreibung im Abschnitt 3.2.2.

Die Abbildung 4.15 zeigt uns die Anzahl der Autoren von BPMN-Dateien. Aus diesem Diagramm kann man ablesen, dass der überwiegende Anteil von allen BPMN-

Dateien (92.1%) nur einen Autor hat. 4,6% der BPMN-Dateien haben zwei Autoren und 2,8% - drei.

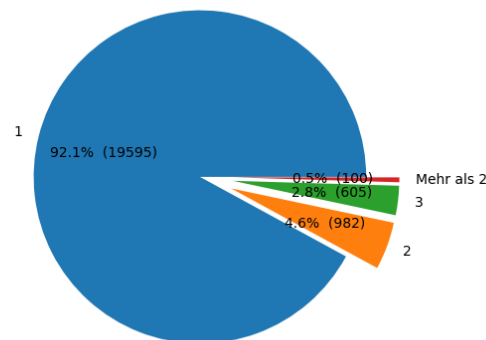


Abbildung 4.15.: Anzahl der Autoren von BPMN-Dateien

#### 4.1.2.5. Dateialter

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_age_n_authors_n_revers()` im Mining-Programm [16] und Beschreibung im Abschnitt 3.2.2.

Die Abbildung 4.16 zeigt uns das Alter von BPMN-Dateien (Zeitraum seit letzter Änderung). Es ist zu sehen, dass die Mehrheit der Dateien (36,1%) nicht älter als 1 Jahr ist. Fast jede vierte Datei ist 1 Jahr alt und jede sechste Datei - 2 Jahre alt. Die älteste BPMN-Datei wurde vor 11 Jahren erstellt. Im Ganzen kann man ablesen, dass seit 5 Jahren jedes Jahr mehr neue BPMN-Dateien erstellt werden, als im Jahr davor.

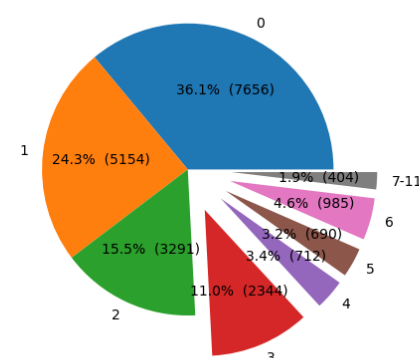


Abbildung 4.16.: Alter in Jahren von BPMN-Dateien

#### 4.1.2.6. Anzahl der Dateiänderungen

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_age_n_authors_n_revers()` im Mining-Programm [16] und Beschrei-

bung im Abschnitt 3.2.2.

Das nächste Merkmal von BPMN-Dateien, das untersucht wurde, war die Anzahl der Dateiänderungen inklusive der Dateierstellung. Wie aus der Abbildung 4.17 zu entnehmen ist, wurden drei Viertel von allen Dateien nach der Erstellung nicht mehr geändert. Der Anteil der Dateien, die zweimal geändert wurden, beträgt 15,9%. Der Anteil der Dateien, die mehr als 2-Mal geändert wurden, beträgt zusammen fast 8%.

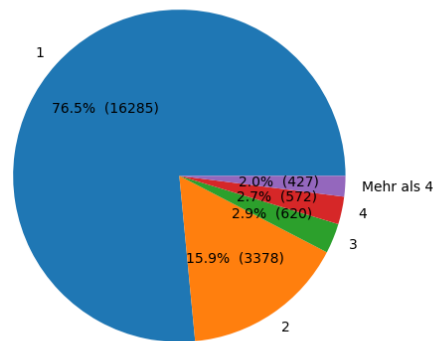


Abbildung 4.17.: Anzahl der Änderungen von BPMN-Dateien

## 4.2. Statistiken über Korrektheit der BPMN-Prozesse im XML-Format

### 4.2.1. Analyse mittels BPMNspector

Wie Matthias Geiger, der Autor von BPMNspector, angibt, werden BPMN-Prozesse häufig mit einigen Verstößen gegen die Syntax- und Semantikregeln von BPMN 2.0 erstellt [9]. Deshalb war es für uns von Interesse, die gefundenen BPMN-Dateien im XML-Format (BPMN-Prozesse) unter Verwendung des Analysewerkzeugs BPMNspector hinsichtlich solcher Verstöße zu untersuchen. Dieses Programm analysiert ein BPMN-Prozess oder ein Verzeichnis mit mehreren BPMN-Prozessen und ermittelt ob BPMN-Prozesse valide sind oder nicht. Falls ein Prozess nicht valide ist, liefert BPMNspector alle Verstöße gegen BPMN 2.0-Einschränkungen, die er hat. Die Liste von allen möglichen Verstößen ist in der Publikation „BPMN 2.0 Process Model Serialization Constraints“ von Matthias Geiger definiert (siehe [17]).

In die Analyse mit BPMNspector wurden 8.904 Dateien von den insgesamt 16.907 gefundenen BPMN-Dateien im XML-Format einbezogen. Bei den restlichen 8.003 Dateien handelt es sich um Kopien dieser Dateien. Siehe Funktionen `create_reports()` und `add_constraints_unfixed()` im Mining-Programm [13] und Beschreibung im Abschnitt 3.2.2.

Wie in der Abbildung 4.18 zu sehen ist, wurden nur 16,5% (1.471) von den untersuchten BPMN-Prozessen als valide Prozesse, das heißt Prozesse ohne Verlet-

zung einer Syntax- oder Semantikregel von BPMN 2.0, eingestuft. In 82,8% (7.376) der untersuchten BPMN-Prozesse wurden verschiedene Verstöße gegen BPMN 2.0-Einschränkungen gefunden. Die restlichen 0,6% (57) BPMN-Prozesse konnten mit dem BPMNspector nicht analysiert werden.

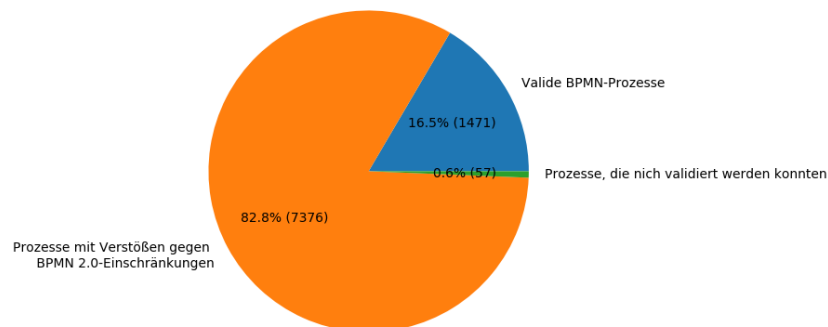


Abbildung 4.18.: Anteil valider und nicht-valider BPMN-Prozesse

Die Abbildung 4.19 zeigt uns eine Statistik zu 7.376 BPMN-Prozessen mit Verstößen gegen BPMN 2.0-Einschränkungen. In dieser Statistik wurden die 10 häufigsten Verstöße ermittelt, wobei das mehrmalige Vorkommen einer Art der Verstöße pro BPMN-Prozess nur einmal gezählt wurde. Es ist zu sehen, dass drei Arten der Verstöße, nämlich „EXT.101“, „EXT.023“ und „EXT.107“ am häufigsten gefunden wurden.

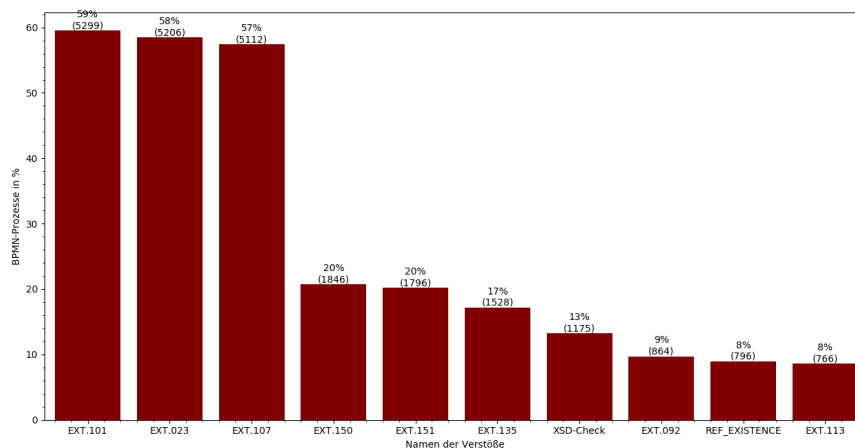


Abbildung 4.19.: BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen (insgesamt 7.376 BPMN-Prozesse)

Der „EXT.101“ Verstoß betrifft das Element Startereignis (StartEvent) eines BPMN-Prozesses. Hier geht es um eine Verletzung der Regel, die besagt, dass „ein Startereignis eine Quelle für einen Sequenzfluss sein muss“ [17].

Der Verstoß „EXT.023“ betrifft den Sequenzfluss (SequenceFlow) in BPMN-Prozessen. Es wird eine Regel verletzt, die besagt, dass „das Quell- und Zielement des Sequenzflusses unter Verwendung der eingehenden oder ausgehenden Attribute auf die SequenceFlow-Definition verweisen muss“ [17] (siehe dazu auch

die Diskussion in Abschnitt 3.2.2 und das Beispiel in Abbildungen 3.7 und 3.8). Wie der Autor von BPMNspector angibt, handelt es sich bei „EXT.023“ um einen der häufigsten Verstöße gegen die BPMN 2.0-Einschränkungen [8]. Unsere Analyse hat diese Aussage vollständig bestätigt. Matthias Geiger verweist auf das von ihm ebenfalls entwickelte Programm BPMNspector-fixSeqFlow [8], das diese Art von Verstößen korrigiert.

Der Verstoß „EXT.107“ betrifft das Element Endereignis (EndEvent). Es geht hier um Verletzung einer Regel, die besagt, dass „ein Endereignis mindestens einen eingehenden Sequenzfluss haben muss“ [17].

Die Bedeutungen von anderen Vorstößen können aus der Tabelle C.1 im Anhang C entnommen werden.

Die Abbildung 4.20 zeigt uns die Top-10 Verstöße gegen BPMN 2.0-Einschränkungen, aufsummiert über alle 7.376 BPMN-Dateien mit Verstößen. Es ist zu sehen, dass der Verstoß „EXT.023“ die größte Anzahl hat, nämlich 58.015 (38,6%). Im Unterschied zur Abbildung 4.19, wo „XSD-Check“ auf dem 7. Platz steht und in 13% der nicht validen BPMN-Prozessen vorkommt, befindet er sich bei der absoluten Zählung mit der Anzahl 43.516 (29%) auf dem zweiten Platz in der Abbildung 4.20. Prozentsätze der anderen Verstöße, die in der Abbildung 4.20 zu sehen sind, sind im Vergleich zu den schon erwähnten deutlich kleiner.

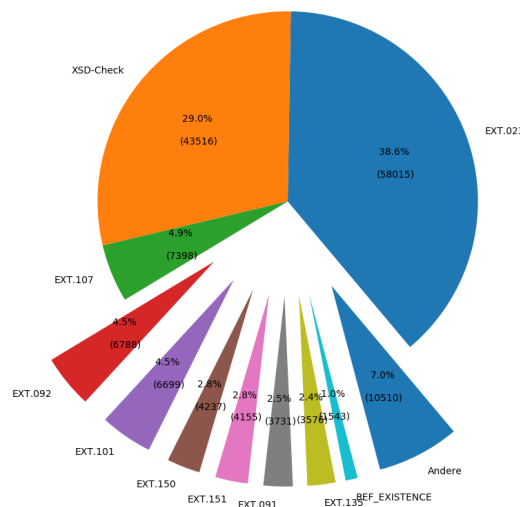


Abbildung 4.20.: Top-10 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 150.168 Verstöße)

Insgesamt hat die Anwendung von BPMNspector auf die gefundenen BPMN-Prozesse im XML-Format gezeigt, dass nur jeder sechste von ihnen in Hinsicht auf die BPMN-Spezifikation ein valider Prozess ist. Daraus kann man schließen, dass ein Bedarf für die Verbesserung der bestehenden BPMN-Werkzeuge existiert, um solche Fehler bei der Modellierung erkennen und beheben zu können. Die Grafiken 4.19 und 4.20 machen deutlich, welche Verstöße gegen BPMN 2.0-Einschränkungen bei der Erstellung der Prozesse berücksichtigt werden müssen.

### 4.2.2. Reparieren mittels BPMNspector-fixSeqFlow

Um die mit dem BPMNspector gefundenen Verstöße gegen BPMN 2.0-Einschränkungen zu reparieren, wurde das BPMNspector-fixSeqFlow Programm angewendet. Siehe Funktionen `fix_bpmn()` und `add_constraints_fixed()` im Mining-Programm [? ], [13] und Beschreibung im Abschnitt 3.2.2. Wie in der Programmbeschreibung steht, repariert BPMNspector-fixSeqFlow den Verstoß „EXT.023“, indem es „die fehlenden <incoming> / <outgoing> -Elemente hinzufügt und eine korrigierte Version eines bestimmten BPMN-Prozesses erstellt“ [8].

Von 8.904 BPMN-Prozesse, an denen BPMNspector-fixSeqFlow ausgeführt wurde, wurden 5.207 geändert. Diese 5.207 BPMN-Prozesse wurden wieder mit dem BPMNspector analysiert. Die Analyse ergab, dass 3.270 BPMN-Prozesse keine Verstöße gegen die Syntax- und Semantikregeln von BPMN 2.0 mehr haben. Das bedeutet, dass sie mit dem BPMNspector-fixSeqFlow repariert wurden. In restlichen 1.937 BPMN-Prozessen hat BPMNspector noch verschiedene Verstöße gefunden.

Der Vergleich der Verstöße vor und nach der Ausführung von BPMNspector-fixSeqFlow hat ergeben, dass es 84.808 Verstöße repariert wurden.

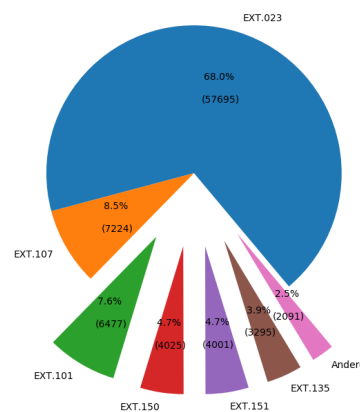


Abbildung 4.21.: Verstöße gegen BPMN 2.0-Einschränkungen, die repariert wurden (insgesamt 84.808 Verstöße)

Die Abbildung 4.22 zeigt die Anteile valider und nicht-valider BPMN-Prozesse nach der Anwendung BPMNspector-fixSeqFlow.

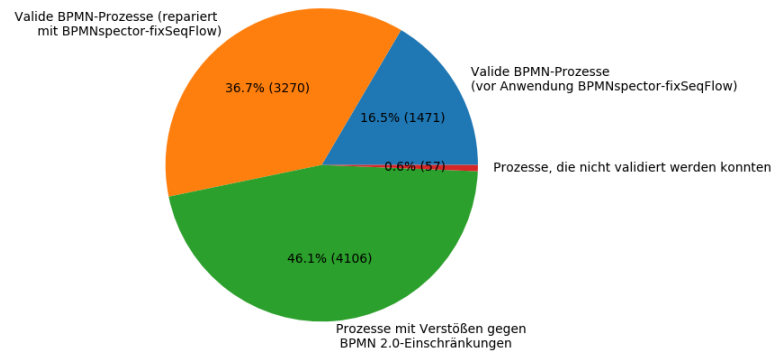


Abbildung 4.22.: Anteil valider und nicht-valider BPMN-Prozesse nach der Anwendung BPMNspectator-fixSeqFlow

Die Abbildung 4.23 zeigt uns eine Statistik zu 4.106 BPMN-Prozessen mit Verstößen gegen BPMN 2.0-Einschränkungen. In dieser Statistik wurden die 10 häufigsten Verstöße ermittelt, wobei das mehrmalige Vorkommen einer Art der Verstöße pro BPMN-Prozess nur einmal gezählt wurde.

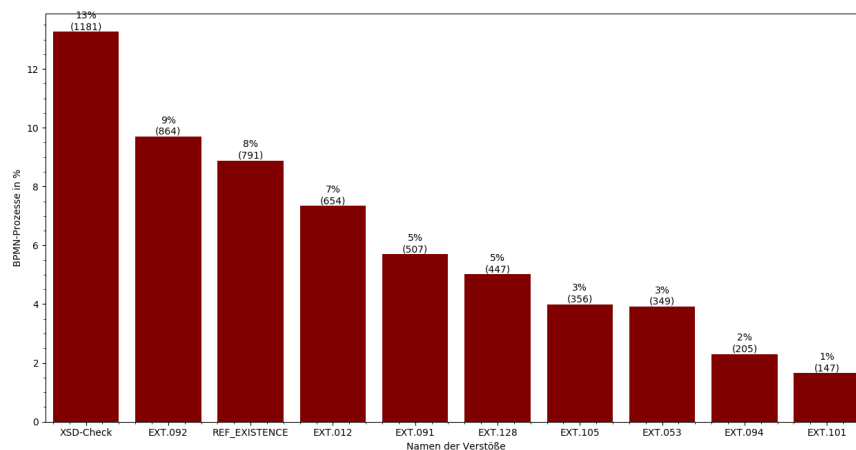


Abbildung 4.23.: BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen nach der Ausführung BPMNspectator-fixSeqFlow (insgesamt 4.106 BPMN-Prozesse)

Die Abbildung 4.24 zeigt uns die Top-5 Verstöße gegen BPMN 2.0-Einschränkungen, aufsummiert über alle 4.106 BPMN-Prozesse mit Verstößen.



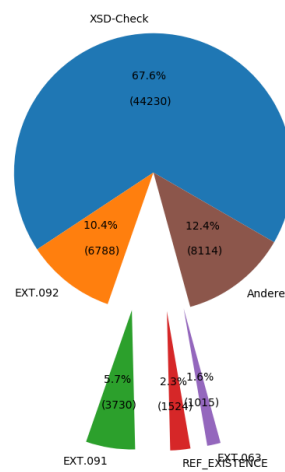


Abbildung 4.24.: Top-5 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 65.401)

## 5. Fazit

Im Rahmen dieser Arbeit wurde das Mining-Programm für die Suche nach BPMN-Prozessen auf GitHub entwickelt. Mit diesem Programm wurden 10% von allen öffentlichen GitHub-Repositorys (6.163.217 GitHub-Repositorys) nach BPMN-Prozessen durchsucht und bei 0,02% von ihnen (1.251 Repositorys) die BPMN-Prozesse gefunden. Diese Repositorys enthalten 21.306 Dateien, die im Namen das Wort „bpmn“ haben. 16.907 von diesen Dateien wurden als BPMN-Prozesse im XML-Format identifiziert.

Die Analyse der GitHub-Repositorys mit BPMN-Prozessen hat uns folgende Erkenntnisse gegeben:

- Die absolute Mehrheit der Repositorys (51,5%) hat Java, als Hauptprogrammiersprache.
- Die Erscheinung der Version 2.0 von BPMN in 2011 war ein großer Schub für die Popularität der BPMN auf GitHub. Der jährliche Zuwachs der GitHub-Repositorys mit BPMN-Prozessen betrug zwischen 2011 und 2017 im Durchschnitt 71,7%.
- China, Vereinigte Staaten und Deutschland wurden als Standorte identifiziert, wo sich die meisten Mitwirkenden von GitHub-Repositorys mit BPMN-Prozessen befinden.

Die wichtigsten Erkenntnisse aus der Analyse von BPMN-Prozessen im XML-Format sind die Folgenden:

- Die absolute Mehrheit der BPMN-Prozesse (91,3%) hat nur einen Autor.
- 38,9% der BPMN-Prozesse sind nicht älter als ein Jahr.
- Die Absolute Mehrheit der BPMN-Prozesse (73%) wurden nach der Erstellung nicht mehr geändert.
- Der Anteil der Duplikate unter den gefundenen BPMN-Prozesse ist relativ groß und beträgt 47,3%.

Die Anwendung des BPMNspectors auf BPMN-Prozesse im XML-Format, ausgenommen reine Kopien, hat ergeben, dass nur 16,5% (1.469) von ihnen valide sind. d.h. ohne Verstöße gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln erstellt wurden. In 83,5% (7.436) der BPMN-Prozesse hat BPMNspector mindestens einen Verstoß gefunden. Aus diesen Zahlen kann man schließen, dass es bei einigen Werkzeugen für die Erstellung der BPMN-Prozesse ein Bedarf existiert, sie in Hinsicht der in der BPMN-Spezifikation definierten Syntax- und Semantikregeln anzupassen.

# Literaturverzeichnis

- [1] Bpmn 2.0-symbolreferenz. <https://camunda.com/bpmn/reference/>. Accessed: 2019-06-24.
- [2] Bpmn-prozess: Expenseprocess. <https://raw.githubusercontent.com/chweich/shopGuns/master/guns-admin/src/main/resources/processes/ExpenseProcess.bpmn20.xml>. Accessed: 2019-07-07.
- [3] Bpmn-prozess: leave. <https://raw.githubusercontent.com/11653786/jeesite/master/src/main/resources/act/designs/oa/leave/leave.bpmn>. Accessed: 2019-07-07.
- [4] Bpmn-prozess: reviewsaleslead. [https://raw.githubusercontent.com/antrad1978/devops/master/openshift\\_activiti6/apache-tomcat-9.0.12/webapps/activiti-rest/WEB-INF/classes/reviewSalesLead.bpmn20.xml](https://raw.githubusercontent.com/antrad1978/devops/master/openshift_activiti6/apache-tomcat-9.0.12/webapps/activiti-rest/WEB-INF/classes/reviewSalesLead.bpmn20.xml). Accessed: 2019-07-07.
- [5] Bpmn-prozess: test\_audit. [https://raw.githubusercontent.com/11653786/jeesite/master/src/main/resources/act/designs/oa/test\\_audit/test\\_audit.bpmn](https://raw.githubusercontent.com/11653786/jeesite/master/src/main/resources/act/designs/oa/test_audit/test_audit.bpmn). Accessed: 2019-07-07.
- [6] Bpmn workflow engine. <https://camunda.com/de/products/bpmn-engine/>. Accessed: 2019-06-21.
- [7] Bpmn\_crawler. [https://github.com/ViktorStefanko/BPMN\\_Crawler/](https://github.com/ViktorStefanko/BPMN_Crawler/). Accessed: 2019-06-28.
- [8] Bpmnspector-fixseqflow. <https://github.com/matthiasgeiger/BPMNspector-fixSeqFlow>. Accessed: 2019-06-11.
- [9] Bpmnspector. static analysis of bpmn 2.0 process models. <http://bpmnspector.org/>. Accessed: 2019-05-12.
- [10] Business process model and notation. <https://www.omg.org/spec/BPMN/2.0.2>. Accessed: 2019-04-15.
- [11] Camunda modeler. <https://camunda.com/de/products/modeler/>. Accessed: 2019-06-21.
- [12] Code maat. <https://github.com/adamtornhill/code-maat>. Accessed: 2019-07-02.
- [13] Constraintschecker. [https://github.com/ViktorStefanko/BPMN\\_Crawler/blob/master/scripts/bpmn\\_spector/run\\_bpmn\\_spector.py](https://github.com/ViktorStefanko/BPMN_Crawler/blob/master/scripts/bpmn_spector/run_bpmn_spector.py). Accessed: 2019-07-07.

- [14] Duplicate files finder. <http://doubles.sourceforge.net/>. Accessed: 2019-05-12.
- [15] FERNANDEZ, M. A., AND ROBLES, G. Extracting software development information from floss projects in github. Paper at SATToSE 2017, [http://sattose.wdfiles.com/local--files/2017:schedule/SATToSE\\_2017\\_paper\\_6.pdf](http://sattose.wdfiles.com/local--files/2017:schedule/SATToSE_2017_paper_6.pdf), 2017. Accessed: 2019-04-20.
- [16] Filestatistics. [https://github.com/ViktorStefanko/BPMN\\_Crawler/blob/master/scripts/statistics/file\\_statistics.py](https://github.com/ViktorStefanko/BPMN_Crawler/blob/master/scripts/statistics/file_statistics.py). Accessed: 2019-07-07.
- [17] GEIGER, M. BPMN 2.0 Process Model Serialization Constraints. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, no. 92, Otto-Friedrich Universität Bamberg, May 2013.
- [18] Ghtorrent on the google cloud. <http://ghtorrent.org/gcloud.html>. Accessed: 2019-04-19.
- [19] Github. <https://github.com/>. Accessed: 2019-04-15.
- [20] Github facts. <https://github.com/about/facts>. Accessed: 2019-04-19.
- [21] Githut 2.0. [https://madnight.github.io/githut/#/pull\\_requests/2018/1](https://madnight.github.io/githut/#/pull_requests/2018/1). Accessed: 2019-07-08.
- [22] Gitlab. <https://about.gitlab.com/>. Accessed: 2019-04-15.
- [23] Google bigquery. <https://cloud.google.com/bigquery/>. Accessed: 2019-04-19.
- [24] GOUSIOS, G., VASILESCU, B., SEREBRENIK, A., AND ZAIDMAN, A. Lean ghtorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (New York, NY, USA, 2014), MSR 2014, ACM, pp. 384–387.
- [25] HASSAN, A. E. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance* (Sep. 2008), pp. 48–57.
- [26] HEBIG, R., QUANG, T. H., CHAUDRON, M. R. V., ROBLES, G., AND FERNANDEZ, M. A. The quest for open source projects that use uml: Mining github. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems* (New York, NY, USA, 2016), MODELS '16, ACM, pp. 173–183.
- [27] HERZIG, K. S., AND ZELLER, A. Mining your own evidence. In *Making Software - What Really Works, and Why We Believe It*. O'Reilly Media, Inc., Sebastopol, 2011, pp. 517–529.
- [28] KALE, V. *Enterprise Process Management Systems - Engineering Process-Centric Enterprise Systems using BPMN 2.0*. CRC Press, Boca Raton, Fla, 2018.

- [29] KOCIAN, C. *Geschäftsprozessmodellierung mit BPMN 2.0 Business Process Model and Notation im Methodenvergleich*. HNU Working Paper Nr. 16. Neu-Ulm University of Applied Sciences, Wileystraße 1, D-89231 Neu-Ulm, 2011.
- [30] Object management group. <https://www.omg.org/>. Accessed: 2019-04-15.
- [31] OMG. *Business Process Model and Notation (BPMN)*. Needham USA(2013), 2013.
- [32] Places with the most contributors. <https://octoverse.github.com/people>. Accessed: 2019-05-22.
- [33] Prozessmodellierung mit epk. <http://www.leed.ch/history/eeepk/>. Accessed: 2019-04-15.
- [34] Repositorystatistics. [https://github.com/ViktorStefanko/BPMN\\_Crawler/blob/master/scripts/statistics/repository\\_statistics.py](https://github.com/ViktorStefanko/BPMN_Crawler/blob/master/scripts/statistics/repository_statistics.py). Accessed: 2019-07-07.
- [35] Rest api v3. <https://developer.github.com/v3/>. Accessed: 2019-04-22.
- [36] ROBLES, G. 2016-uml-miner. <https://github.com/LibreSoftTeam/2016-uml-miner>. Accessed: 2019-04-23.
- [37] ROBLES, G., HO-QUANG, T., HEBIG, R., CHAUDRON, M., AND FERNANDEZ, M. An extensive dataset of uml models in github. pp. 519–522.
- [38] ROBLES, G., HO-QUANG, T., HEBIG, R., CHAUDRON, M. R. V., AND FERNANDEZ, M. A. An extensive dataset of uml models in github. In *Proceedings of the 14th International Conference on Mining Software Repositories* (Piscataway, NJ, USA, 2017), MSR '17, IEEE Press, pp. 519–522.
- [39] SANTACROCE, F., OLSSON, A., VOSS, R., AND NAREBSKI, J. *Git: Mastering Version Control*. Packt Publishing Ltd, Birmingham, 2016.
- [40] The ghtorrent project. <http://ghtorrent.org/>. Accessed: 2019-04-19.
- [41] The state of the octoverse. <https://octoverse.github.com/>. Accessed: 2019-04-19.
- [42] TORNHILL, A. *Your Code as a Crime Scene - Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs*. Pragmatic Bookshelf, Raleigh, North Carolina and Dallas, Texas, 2015.
- [43] Übersicht zur prozessmodellierungssprache bpmn 2.0. <https://www.processhub.com/uebersicht-zur-prozessmodellierungssprache-bpmn-2-0/>. Accessed: 2019-06-07.
- [44] Unified modeling language. <https://www.uml.org/>. Accessed: 2019-04-15.

# Abbildungsverzeichnis

2.1. Camunda Modeler (grafischer Editor) . . . . .	9
2.2. Camunda Modeler (Texteditor) . . . . .	9
2.3. Buchausleihe . . . . .	14
2.4. Algorithmus für das MSR . . . . .	16
3.1. MySQL-Datenbankschema der Tabellen „users“ und „projects“ . . .	19
3.2. Beispiel von einem zurückgelieferten JSON-Objekt von Suchschnitt- stelle der GitHub API . . . . .	21
3.3. Datenbankschema . . . . .	23
3.4. Prozessdiagramm des Mining-Programms . . . . .	25
3.5. Klassendiagramm des Mining-Programms . . . . .	26
3.6. Ein einfacher BPMN-Prozess . . . . .	30
3.7. Ein einfacher BPMN-Prozess mit Verletzung einer <sequenceFlow> Einschränkung. Übernommen aus [8]. . . . .	30
3.8. Ein einfacher BPMN-Prozess ohne Verletzung einer <sequence- Flow> Einschränkung. Übernommen aus [8]. . . . .	30
3.9. Dateiformate von potentiellen BPMN-Prozessen (gruppiert) . . . . .	31
4.1. Dominante Programmiersprachen der Repositorys mit BPMN-Dateien	34
4.2. Dominante Programmiersprachen von allen öffentlichen GitHub- Repositorys, erstellt im 1. Quartal 2018 [21] . . . . .	34
4.3. Jahre, in denen Repositorys mit BPMN-Dateien erstellt wurden . . .	35
4.4. Jahre, in denen Repositorys mit BPMN-Dateien letztes Mal geändert wurden . . . . .	35
4.5. Jahre, in denen Repositorys mit BPMN-Dateien erstellt und letztes Mal geändert wurden . . . . .	36
4.6. Dauer der Aktivitäten von GitHub-Repositorys mit BPMN-Dateien (Kreisdiagramm) . . . . .	37
4.7. Dauer der Aktivitäten von GitHub-Repositorys mit BPMN-Dateien (Balkendiagramm) . . . . .	37
4.8. Anzahl der Commits von GitHub-Repositorys mit BPMN-Dateien . .	38
4.9. Standorte der Repository-Entwickler . . . . .	38
4.10. Standorte mit den meisten GitHub-Benutzer [32] . . . . .	39
4.11. Angaben über Duplikate von BPMN-Dateien . . . . .	40
4.12. Beispiel eines BPMN-Prozess im XML-Format . . . . .	41
4.13. Anzahl der BPMN-Prozesse im XML-Format mit dem XML-Element „BPMNDiagramm“ . . . . .	41
4.14. Anzahl der XML-Elemente von BPMN-Dateien im XML-Format . . .	42

4.15. Anzahl der Autoren von BPMN-Dateien . . . . .	43
4.16. Alter in Jahren von BPMN-Dateien . . . . .	43
4.17. Anzahl der Änderungen von BPMN-Dateien . . . . .	44
4.18. Anteil valider und nicht-valider BPMN-Prozesse . . . . .	45
4.19. BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen (insgesamt 7.376 BPMN-Prozesse) . . . . .	45
4.20. Top-10 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 150.168 Verstöße) . . . . .	46
4.21. Verstöße gegen BPMN 2.0-Einschränkungen, die repariert wurden (insgesamt 84.808 Verstöße) . . . . .	47
4.22. Anteil valider und nicht-valider BPMN-Prozesse nach der Anwen- dung BPMNspector-fixSeqFlow . . . . .	48
4.23. BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen nach der Ausführung BPMNspector-fixSeqFlow (insgesamt 4.106 BPMN-Prozesse) . . . . .	48
4.24. Top-5 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 65.401)	49
A.1. BPMN-Prozess [5], der 68-Mal dupliziert wurde . . . . .	57
A.2. BPMN-Prozess [3], der 61-Mal dupliziert wurde . . . . .	57
A.3. BPMN-Prozess [4], der 29-Mal dupliziert wurde . . . . .	58
A.4. BPMN-Prozess [2], der 28-Mal dupliziert wurde . . . . .	58
B.1. 2716 . . . . .	60
B.2. 2795 . . . . .	61
B.3. 2847 . . . . .	62
B.4. 3012 . . . . .	63
B.5. 3347 . . . . .	64

# Tabellenverzeichnis

2.1. Grundlegende Modellierungselemente von BPMN-Prozessen [10], [29], [1] . . . . .	10
2.1. Grundlegende Modellierungselemente von BPMN-Prozessen . . . .	11
2.1. Grundlegende Modellierungselemente von BPMN-Prozessen . . . .	12
2.1. Grundlegende Modellierungselemente von BPMN-Prozessen . . . .	13
3.1. Resultate des Mining-Programms . . . . .	32
C.1. Verstöße gegen BPMN 2.0-Einschränkungen . . . . .	65
C.1. Verstöße gegen BPMN 2.0-Einschränkungen . . . . .	66



## A. Anhang

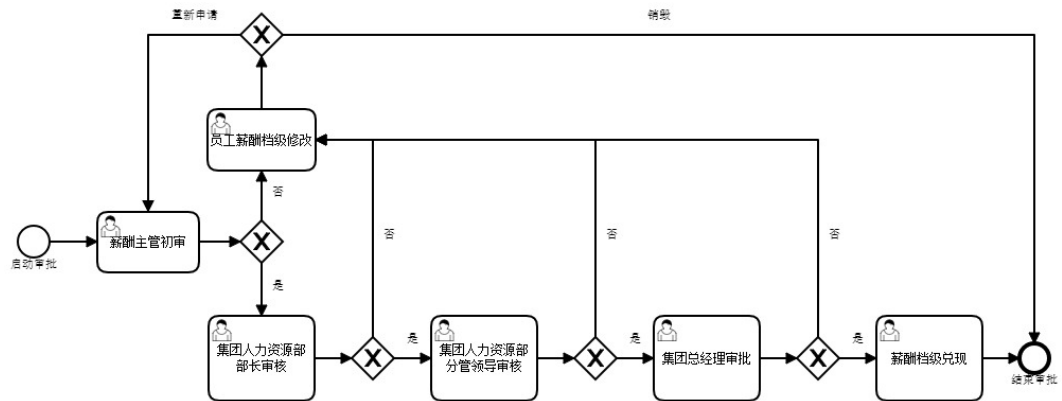


Abbildung A.1.: BPMN-Prozess [5], der 68-Mal dupliziert wurde

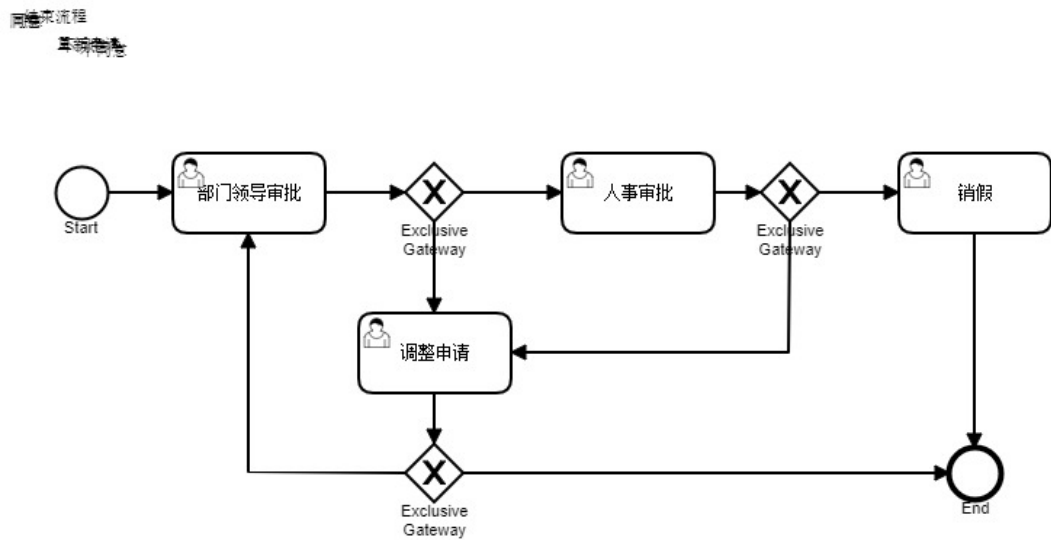


Abbildung A.2.: BPMN-Prozess [3], der 61-Mal dupliziert wurde

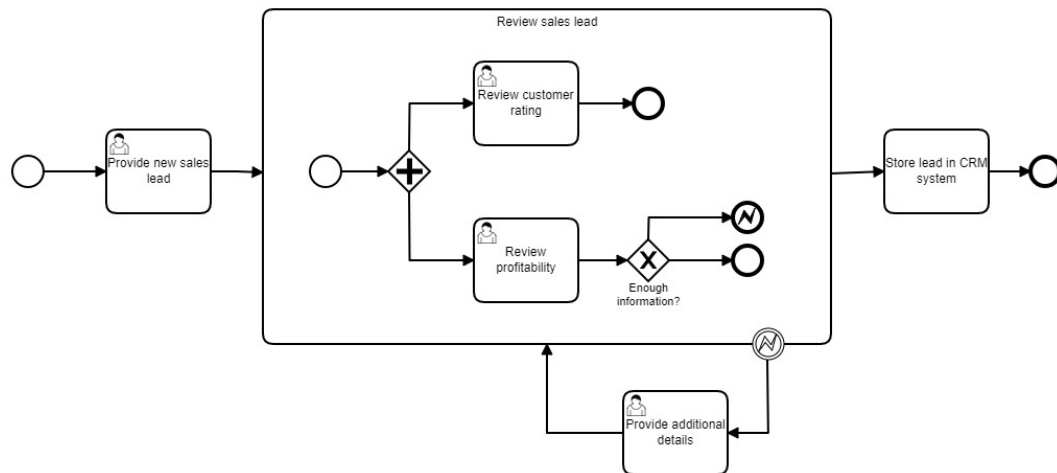


Abbildung A.3.: BPMN-Prozess [4], der 29-Mal dupliziert wurde

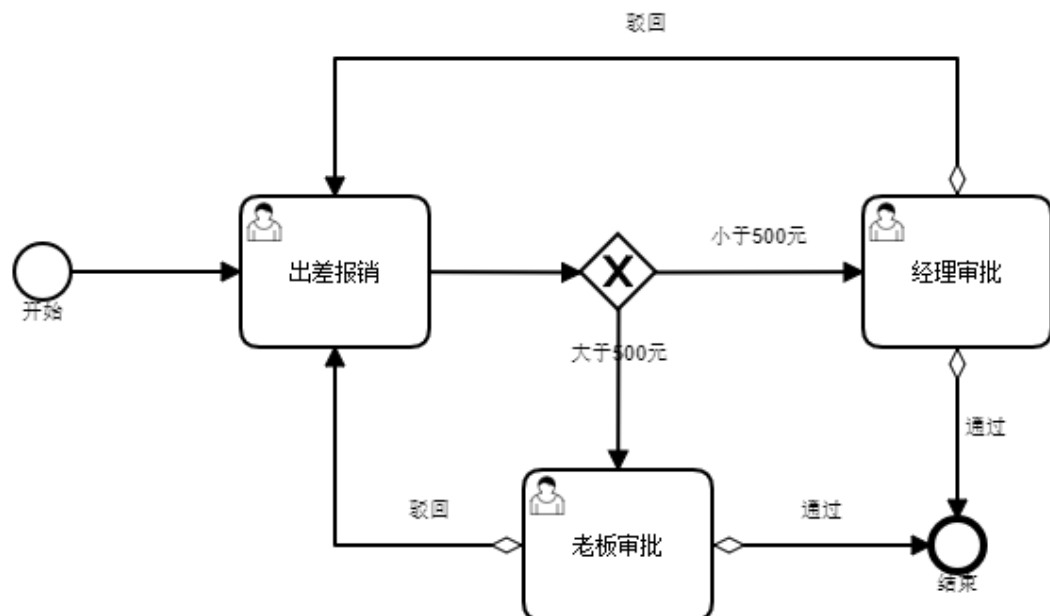


Abbildung A.4.: BPMN-Prozess [2], der 28-Mal dupliziert wurde

## **B. Anhang**

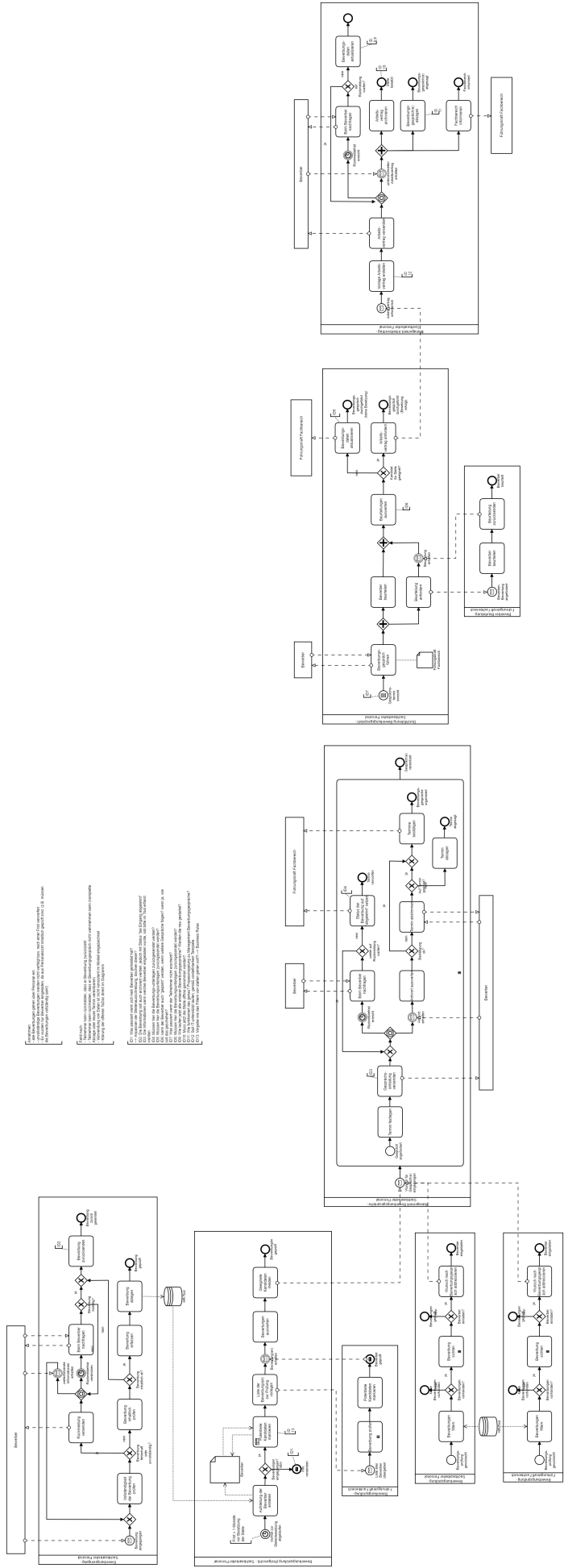


Abbildung B.1.: 2716

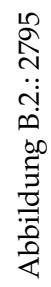
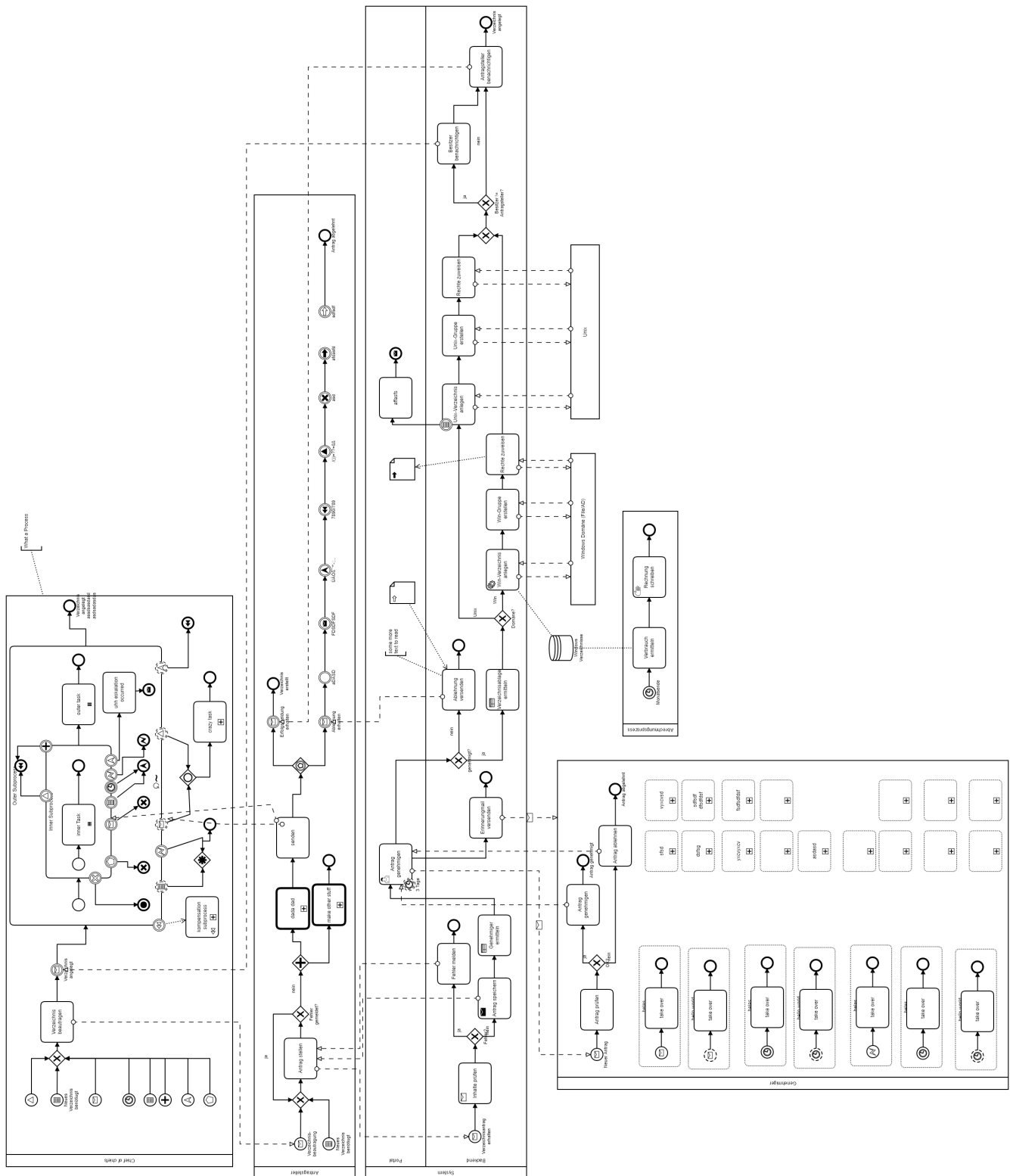


Abbildung B.2.: 2795



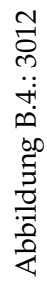


Abbildung B.4.: 3012

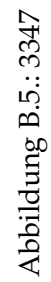


Abbildung B.5.: 3347



## C. Anhang

Tabelle C.1.: Verstöße gegen BPMN 2.0-Einschränkungen

	Name des Verstoßes	Betroffenes Element	Bedeutung
1	EXT.012	<Expression>	Wenn Ausdrücke in natürlicher Sprache verwendet werden, kann der Prozess nicht ausgeführt werden. [17].
2	EXT.023	<Sequence Flow>	Das Quell- und Zielement des Sequenzflusses müssen unter Verwendung der eingehenden oder ausgehenden Attribute auf die SequenceFlow-Definition verweisen [17].
3	EXT.013	<Formal Expression>	Formeller Rumpf muss vorhanden sein [17].
4	EXT.053	<Script Task>	Wenn ein Skript vorhanden ist, muss der Skripttyp definiert werden [17].
5	EXT.067	<Start Event>	Wenn keine Transformation definiert oder referenziert ist, MUSS nur eine Quelle definiert werden. [17].
6	EXT.092	<Data Association>	Wenn keine Transformation definiert oder referenziert ist, MUSS nur eine Quelle definiert werden. [17].
7	EXT.094	<Event>	Für jede Ereignisdefinition muss eine Elementdefinition mit der zugehörigen Dateneingabe / -ausgabe vorhanden sein [17].
8	EXT.101	<Start Event>	Der Startereignis muss die Quelle für den Sequenzfluss sein [17].
9	EXT.105	<End Event>	Ein Endereignis muss vorhanden sein, wenn ein Startereignis auf derselben Prozessebene verwendet wird. [17].
10	EXT.107	<End Event>	Das Endereignis muss mindestens einen eingehenden Sequenzfluss haben [17].
11	EXT.113	<Boundary Event>	Ein Grenzergebnis muss mindestens eine Quelle für einen Sequenzfluss sein [17].

Tabelle C.1.: Verstöße gegen BPMN 2.0-Einschränkungen

	<b>Name des Versto- ßes</b>	<b>Betroffenes Element</b>	<b>Bedeutung</b>
12	<b>EXT.128</b>	<Message Event De- finition>	Ein operationRef muss vorhanden sein, wenn der Prozess ausführbar sein soll. [17].
13	<b>EXT.135</b>	<Gateway>	Ein Gateway muss entweder mehrere eingehende Sequenzflüsse oder mehrere ausgehende Sequenzflüsse haben [17].
14	<b>EXT.150</b>	<Start Event>	Wenn ein Startereignis zum Initiieren eines Prozesses verwendet wird, müssen alle Ablaufknoten (außer Startereignissen, Grenzergebnissen und abfangenden Verknüpfungsergebnissen, Kompensationsaktivitäten und Ereignis-Unterprozessen) einen eingehenden Ablauf haben [17].
15	<b>EXT.151</b>	<End Event>	Wenn Endereignisse verwendet werden, müssen alle anderen Ablaufknoten (außer Endergebnissen, auslösenden Verknüpfungsergebnissen, Ausgleichsaktivitäten und Ereignisunterprozessen) einen ausgehenden Ablauf haben [17].
16	<b>XSD- Check</b>	-	Fehler in Dateistruktur.
17	<b>REF_EX ISTEN CE</b>	-	Wenn ein BPMN-Element auf ein anderes Element verweist, muss die Referenz auflösbar sein. D.h. das referenzierte Element muss vorhanden sein. Darüber hinaus dürfen in den meisten Fällen nur bestimmte Elemente referenziert werden [17].